# Table of Contents

1 Installation........................................................................................................................................... 6

1.1 INSTALLATION OVERVIEW.............................................................................................................. 6
1.2 INTERPRETER VS COMPILER............................................................................................................. 6
1.3 REQUIRED SYSTEM SOFTWARE......................................................................................................... 6
1.4 BASIC SOFTWARE INSTALLATION.................................................................................................... 6
1.5 POSTGRESQL SOFTWARE.................................................................................................................. 7
1.6 MySQL SOFTWARE............................................................................................................................ 7
1.7 Windows (Cygwin) SOFTWARE........................................................................................................... 7
1.8 DATABASE INSTALLATION.................................................................................................................. 7
   1.8.1 Native B-tree Database................................................................................................................... 9
   1.8.1.1 Native B-tree Client-Server Database......................................................................................... 9
   1.8.1.2 Native B-tree Database (Standalone).......................................................................................... 10
   1.8.2 Relational Database Server Resident Global Arrays..................................................................... 10
   1.8.2.1 Overview.................................................................................................................................. 10
   1.8.2.2 Basic Database Configuration..................................................................................................... 12
   1.8.2.3 Relational Database Configure Options Common to All Servers........................................... 12
       1.8.2.3.1 --with-datasize=numeric-value......................................................................................... 12
       1.8.2.3.2 --with-d dbname= name...................................................................................................... 12
       1.8.2.3.3 --with-indexsize=numeric-value....................................................................................... 12
       1.8.2.3.4 --with-tabsize=number..................................................................................................... 13
   1.8.2.4 Initialization of a Mumps Relational Database.......................................................................... 13
   1.8.3 Relational Database Server Connection for Mumps Global Arrays........................................... 13
   1.8.4 MySQL Database Option.............................................................................................................. 13
   1.8.4.1 MySQL Manual Installation....................................................................................................... 14
   1.8.4.2 MySQL Installation Options..................................................................................................... 14
   1.8.4.3 Mumps Build for MySQL Resident Global Arrays.................................................................. 15
   1.8.4.4 Configuring a Remote MySQL Server...................................................................................... 15
   1.8.4.5 Using MySQL Resident Global Arrays from Mumps.............................................................. 15
   1.8.4.6 Command Line Interpreter mysql.......................................................................................... 15
   1.8.5 PostgreSQL Database Option....................................................................................................... 15
   1.8.5.1 PostgreSQL Options................................................................................................................ 16
   1.8.5.2 Command Line Interpreter psql................................................................................................ 16
   1.8.5.3 Installing PostgreSQL.............................................................................................................. 16
       1.8.5.3.1 Quick PostgreSQL Installation.......................................................................................... 16
   1.8.5.4 PostgreSQL Configuration........................................................................................................ 17
   1.8.5.5 PostgreSQL Specific Mumps Install Options............................................................................ 17
       1.8.5.5.1 Configuring the listen_address.......................................................................................... 18
       1.8.5.5.2 Performance Tuning........................................................................................................... 19
       1.8.5.5.3 PostgreSQL Server Connections....................................................................................... 20

1.9 PostgreSQL TRANSACTION LIMIT.................................................................................................. 20

1.10 USE WITH APACHE............................................................................................................................ 20

1.11 USE WITH WINDOWS......................................................................................................................... 20

1.11.1 Windows/Cygwin Install............................................................................................................. 21
1.12 Math Options.................................................................21
1.13 Numeric Configuration Options.......................................21
1.14 All Configure Options....................................................22

2 Running a Mumps Program..................................................25
2.1 Start the Global Array Server.............................................25
2.2 Mumps CLI Interpreter....................................................25
  2.2.1 Mumps CLI Special Commands.
      2.2.1.1 'globals
      2.2.1.2 'halt 'quit 'halt 'q
      2.2.1.3 'sys cmd
      2.2.1.4 'mumps pgm
      2.2.1.5 'sql sqlCommand
  2.3 Mumps Programs (scripts)

3 Relational Database Commands & Variables..........................28
3.1 Creating Global Array Relational Database Tables................28
3.2 Mumps Access to Relational Tables Not Created by Mumps....28
3.3 SQL Commands in Mumps................................................30
   3.3.1 sql string
   3.3.2 sql/c SQL Disconnect...
   3.3.3 sql/f Format SQL Table
   3.3.4 Added BuiltIn SQL Variables
      3.3.4.1 $zsql
      3.3.4.2 $zsqlOpen
      3.3.4.3 $znative
      3.3.4.4 $zmysql
      3.3.4.5 $zpostgres
      3.3.4.6 $table
      3.3.4.7 $tabsize
      3.3.4.8 $zsqlOutput
      3.3.4.9 %globals()
      3.3.5 SQL Command Output

4 Mixing Mumps and SQL Code.................................................32

5 Implementation Notes.......................................................36
5.1 Notes on Arithmetic Precision.........................................36
   5.1.1 $fnumber
   5.1.2 Exponential format numbers
   5.1.3 Arithmetic Precision
      5.1.3.1 Floating Point Precision
      5.1.3.2 Integer Precision
      5.1.3.3 Performance
   5.1.4 Rounding

5.2 Lock Command with PostgreSQL or MySQL.......................36
5.3 Lock Command in Client/Server Mode.................................36
5.4 Line Continuation..........................................................37
5.5 Naked Indicator............................................................37
5.6 Job Command...............................................................37
5.7 File Names Containing Directory Information....................37
5.8 File Names.................................................................37
5.9 Array Index Collating Sequence.......................................37
5.10 Subroutine & Function Calls..........................................38
   5.11 $fNUMBER() Function................................................39
   5.12 $SELECT() Function....................................................39
   5.13 Compiling Large Programs..........................................39
   5.14 Embedded Expressions................................................39

6 Shell Command....................................................................41
8 Z Functions and System Variables

8.1 System Variables
8.1.1 $ztable
8.1.2 $zTabSize
8.1.3 $zProgram

8.2 Math Functions
8.2.1 $zabs(arg) absolute value
8.2.2 $zacos(arg) arc cosine
8.2.3 $zasin(arg) Arc sine
8.2.4 $zatan(arg) Arc tangent
8.2.5 $zcos(arg) Cosine
8.2.6 $zexp(arg) Exponential
8.2.7 $zexp2(arg) Exponential base 2
8.2.8 $zexp10(arg) Exponential base 10
8.2.9 $zlog(arg) Natural log
8.2.10 $zlog2(arg) Base 2 log
8.2.11 $zlog10(arg) Base 10 log
8.2.12 $zpow(arg1,arg2) Power function
8.2.13 $zsqrt(arg) Square root
8.2.14 $zsqr(arg) Sine function
8.2.15 $ztan(arg) Tangent function

8.3 Date Functions
8.3.1 $zdate(or $zd) formatted date string
8.3.2 $zd1 numeric internal date
8.3.3 $zd2(InternalDate) date conversion
8.3.4 $zd3(Year,Month,Day) Julian date
8.3.5 $zd4(Year,DayOfYear) Julian to Gregorian
8.3.6 $zd5(Year,Month,Day) comma listed date
8.3.7 $zd6 hour:minute
8.3.8 $zd7 hyphenated date
8.3.9 $zd8 hyphenated date with time

8.4 Special Purpose Functions
8.4.1 $zb(arg) remove blanks
8.4.2 $zchdir(directory_path) change directory
8.4.3 $zCurrentFile Current Mumps File
8.4.4 $zdump([filename]) dump global arrays
8.4.5 $zrestore([arg]) restore globals
8.4.6 $zfile(arg) file exists test
8.4.7 $zflush flush Btree buffers
8.4.8 $zgetenv(arg) get environment variable
8.4.9 $zhtml(arg) encode HTML string
8.4.10 $zhit global array cache hit ratio
8.4.11 $zlower(string) convert to lower case
8.4.12 $znormal([arg1],[arg2]) word normalization
8.4.13 $zNoBlanks(arg) remove all blanks
8.4.14 $zpad(arg1,arg2) left justify with padding
8.4.15 $zseek(arg)
8.4.16 $zsrand(arg)
8.4.17 $zstem(arg)
8.4.18 $zsystem(arg)
8.4.19 $ztell
8.4.20 $zu(expression)
8.4.21 $zwi(arg)
8.4.22 $zwn extract words from buffer ................................................................. 48
8.4.23 $zwp extract words from buffer ................................................................. 48
8.4.24 $zws(string) initialize internal buffer ......................................................... 48
8.4.25 Scan Functions ......................................................................................... 48
  8.4.25.1.1 $zZScan .......................................................... 48
  8.4.25.1.2 $zZScanALnum .................................................. 48
  8.4.25.1.3 $zZInput(var) ................................................... 48

8.5 Vector and Matrix Functions ........................................................................... 50
  8.5.1 $ZzAvg(vector) ......................................................................................... 50
  8.5.2 $ZzCentroid(gblMatrix, gblRef) ............................................................. 50
  8.5.3 $ZzCount(gblMatrix) .............................................................. 50
  8.5.4 $ZzMax(gbl) ......................................................................................... 51
  8.5.5 $ZzMin(gbl) ......................................................................................... 51
  8.5.6 $ZzMultiply(gbl1, gbl2, gbl3) ................................................................. 51
  8.5.7 $zZSum(gblMatrix) .............................................................. 51
  8.5.8 $ZzTranspose(gblMatrix1, gblMatrix2) ................................................. 51

8.6 Text Processing Functions ............................................................................. 51
  8.6.1 Similarity Functions ............................................................... 52
    8.6.1.1 $ZzCosine(gbl1, gbl2) ....................................................... 52
    8.6.1.2 $ZzSim1(gbl1, gbl2) ......................................................... 52
    8.6.1.3 $ZzDice(gbl1, gbl2) .......................................................... 52
    8.6.1.4 $ZzJaccard(gbl1, gbl2) ...................................................... 52
  8.6.2 $ZzBMGSearch(arg1, arg2) ................................................................. 53
  8.6.3 $ZzPerlMatch(string, pattern) ............................................................. 53
  8.6.4 $ZzReplace(string, pattern, replacement) .......................................... 55
  8.6.5 $ZzShred(string, length) ................................................................. 55
  8.6.6 $ZzShredQuery(string, length) ........................................................... 55
  8.6.7 $ZzSoundex(s1) ................................................................................. 57
  8.6.8 $ZzSmithWaterman(s1, s2, align, mat, gap, noMatch, match) .......... 57
  8.6.9 $ZzIDF(global, doccount) ................................................................. 57
  8.6.10 Correlation Functions ........................................................................... 58
    8.6.10.1 $ZzTermCorrelate(global1, global2) ........................................ 58
    8.6.10.2 $ZzDocCorrelate(gblref1, gblref2, mthd, thrshld) .................. 59
  8.6.11 Stop and Synonym Functions .............................................................. 59
    8.6.11.1 $ZzStopInit(arg) ................................................................. 59
    8.6.11.2 $ZzStopLookup(word) ............................................................ 59
    8.6.11.3 $ZzSynInit(fileName) ............................................................... 59
    8.6.11.4 $ZzSynLookup(word) ............................................................... 59

8.7 SQL Functions ......................................................................................... 60
  8.7.1 $zsql .......................................................... 60
  8.7.2 $zsqlCols .......................................................... 61
  8.7.3 $zsqlOpen .......................................................... 61
  8.7.4 $zNative .......................................................... 61
  8.7.5 $zMysql .......................................................... 61
  8.7.6 $zPostgres .......................................................... 61
  8.7.6.1 $zTable .......................................................... 61
  8.7.7 $zTabsize .......................................................... 61
  8.7.8 $zsqlOutput .......................................................... 61

9 Pattern Matching ..................................................................................... 63
  9.1 Mumps 95 Pattern Matching ................................................................. 63
  9.2 Using Perl Regular Expressions ............................................................. 63

Index of Figures

Figure 1 Creating a View .................................................................................... 29
Figure 2 Mumps View Access ........................................................................... 29
Figure 3 View with Data Conversion .............................................................. 29
Figure 4 Mixing Mumps and SQL ................................................................. 32
Figure 5 Expensive Iterative Mumps Code .................................................... 32
1 Installation

1.1 Installation Overview

1.2 Interpreter vs Compiler

Please do not use the Mumps compiler. It has not been updated and there are possible errors. If you insist upon using it, do not send error reports. It will be brought up to date in a later release.

Use the Mumps interpreter instead.

1.3 Required System Software

Building mumps requires that your system have certain software installed. For the most part, these are available through the Synaptic Package Manager:

1. Linux, preferably a Debian based version such as Debian, Ubuntu or Mint.

2. The g++/gcc compilers and related libraries.

3. The pcre (Perl Compatible Regular Expression) development libraries. The pcre libs should be in /usr/lib and the include files in /usr/include. Be certain to install the pcre development libraries.

4. PostgreSQL and/or MySQL (optional) to store global arrays. Installation must include the client development libraries.

5. The bash shell interpreter located in /bin.

6. The GNU readline and readline-dev packages.

7. Autoconf

8. The following libraries are needed for the extended precision mathematics. If they are not installed by default, you will need to do so. Be sure to install the development versions of the libraries:

   a) The GNU Multiple precision floating point computation library
      http://www.mpfr.org/
      libmpfr-dev

   b) The GNU Multiprecision arithmetic library development tools
      https://gmplib.org/
      libgmp-dev

9. Cygwin DLLs (see below) for use with Windows.

1.4 Basic Software Installation

The following installation are the apt-get tool for the Debian GNU/Linux and related distributions (such as Ubuntu and Mint). Other Linux systems use different but similar tools. You need to install these packages for all versions of Mumps:

apt-get -q -y install autoconf
apt-get install libreadline6 libreadline6-dev
apt-get -q -y install libpcre3
apt-get -q -y install libpcre3-dev

---

1 Note: these are automatically installed if you use mysqlConfigure.script, nativeConfigure.script, nativeClientServerConfigure.script, or postgressConfigure.script -- see below for details.
apt-get -q -y install g++
apt-get -q -y install gcc-doc

apt-get -q -y install libgmp-dev
apt-get -q -y install libmpfr-dev

1.5 PostgreSQL Software
Install these packages if you will be using PostgreSQL to store the global arrays. These are automatically installed if you use \textit{postgressConfigure.script}.

```
VERSION=9.3
apt-get update
apt-get -y install postgresql
apt-get -y install libpq-dev
apt-get -y install libpq5
apt-get -y install postgresql-doc-$VERSION
```

1.6 MySQL Software
Install these packages if you will be using MySQL to store the global arrays. These are automatically installed if you use \textit{mysqlConfigure.script}.

```
apt-get -q -y install mysql-client
apt-get -q -y install mysql-server
apt-get -q -y install mysql-common
apt-get -q -y install mysql-utilities
apt-get -q -y install mysql-workbench
apt-get -q -y install libmysqlclient-dev
apt-get -q -y install libmysqlud-dev
```

1.7 Windows (Cygwin) Software
These DLLs are required for the Cygwin for Windows version (see section 1.11).

```
cygcrypto-1.0.0.dll
cygpcre-1.dll
cygstdc++-6.dll
cygz.dll
cygmysqlclient-18.dll
cygssl-1.0.0.dll
cygwin1.dll
cyggcc_s-1.dll
```

1.8 Database Installation
Installation involves compiling and linking the source code modules. The \textit{configure} program tailors the code to set a number of options most of which you will probably not change (see section 1.14 on page 22 for a complete list of options). Included in the distribution are a set of \textit{bash} scripts that configure and build Mumps.

This Mumps distribution has four options with regard to storing the global arrays:

1. Store the global arrays in the native B-tree database.
   a) single user version
   b) multi-user client-server version
2. Store the global arrays in a local or remote PostgreSQL database.
3. Store the global arrays in a local or remote MySQL database.

Option 1, referred to as the native database, is quite fast with a minimum of overhead and it can efficiently manage very large databases but it lacks a number of features normally found on modern
database systems. It is sensitive to system and programming errors. It does a minimum of
checkpointing and maintains a large part of the global array tree in volatile memory. If the host
system crashes or the program using the global arrays terminates unexpectedly, the contents of
the entire global array database are likely to be lost.

However, in applications where speed is important and, in the event of a crash, the program can
be re-run without loss of data, the native database is a good choice.

The native database has two configurations. The first of these is a single user global array facility
where the global arrays are stored in one directory, usually the one in which the Mumps program is
itself running. In this mode, only one read-write² Mumps program may access the global arrays in a
given directory at a time although other Mumps programs may run concurrently in other directories
operating on other global array data sets. This is the fastest but most restrictive option.

A second native database configures involves running a local global array server. The server
accepts global array access requests by means of internal operating system pipes from one or more
Mumps client programs operating concurrently. This option is slightly slower (about 30%) due to
additional system overhead but it permits multiple concurrent program execution. Only clients
running on the same machine as the server, however, may access the database as there is no
networking option. The server's global array files are slightly better protected as the server is not
affected by client program crashes and the server does periodic flushing of it's buffers and
journaling is possible. However, synchronization of array access is more tricky and requires use of
the awkward and error prone Mumps lock command. Also, the global array database may be lost
due to host system crash or failure of the Mumps server.

If data integrity, remote and multi-user access are important, options 2 and 3 are better. These
use PostgreSQL and MySQL, respectively, to store the global arrays. However, while the global
array access is slower there are several other advantages.

While options 2 and 3 are slower than option 1, due to relational data base system overhead,
using a relational database has significant advantages with regard to reliability and flexibility. These
include:

1. All database transactions are ACID (Atomicity, Consistency, Isolation, Durability) compliant.
2. SQL commands such as Begin Transaction, Commit and Rollback are available.
3. The Mumps global arrays can be queried with SQL commands from non-Mumps environments.
4. SQL views of the Mumps database may be constructed.
5. The Mumps global array database can be remote and distributed.
6. Mumps programs can execute SQL commands on the server on any accessible database table.
7. Multiple concurrent Mumps programs may run at the same time.

The distribution contains several scripts that will build various versions of the system. These are
detailed next. You must be root to run these.

The scripts assume a Debian (apt-get) based Linux installation. If you are using a version of Linux
not based on Debian, you will need to install and configure the required system software manually
according to the procedures on your system.

Some of the scripts provided with the distribution may install system software as needed.
Consequently, when using these scripts, your machine needs to have a reliable Internet connection.
Also, due to Internet load factors, it is possible that software installations may take a long time or, in
some cases, fail in the unlikely event that the servers from which the software to be downloaded are
unavailable.

The Mumps interpreters and libraries built as a result of the scripts will be stored in /usr/bin,
/usr/lib, /usr/include and, in the case of the native file system server code, /etc/mumps.

² The native database Mumps comes in two versions: a read-write version which may both read and write
global arrays and an read-only version where each Mumps program may only read the global arrays. Multiple
read-only instances may operate concurrently on the same global array data sets.
For the most part, except for the Cygwin build, there are two scripts for each option. One of these installs all necessary system software and then builds the Mumps interpreter. The second only configures and builds Mumps, it does not install system software.

In the case of the scripts used with the relational databases, you will need to know the root password for the relational system if it is already installed or, if the script installs it, you will need to give the relational system a root password. In the case of the relational databases, the scripts will create a user named `mumps` and a database named `mumps`. The `mumps` user is granted superuser rights.

### 1.8.1 Native B-tree Database

#### 1.8.1.1 Native B-tree Client-Server Database

`nativeClientServerConfigure.script`

`BuildMumpsWithNativeClientServer.script`

These build the client-server native database global array file system.

The script ultimately create two binary executable programs: `mumpsd`, which is the database server, and `mumps`, which is the interpreter client. See below for details.

The first script, `nativeClientServerConfigure.script`, will install any missing system software and then invoke the second script which actually compiles and installs the Mumps programs.

If required system software is already installed and you only use the second script, `BuildMumpsWithNativeClientServer.script`

If required system software is not installed, you must install the software manually or use the first script.

The executable Mumps client interpreter is place in `/usr/bin` while the Mumps global array server is placed in `/etc/mumps`.

The Mumps global array server permits multiple, local Mumps client programs to access the global arrays concurrently. There is approximately a 30% performance penalty on global array accesses through the server as opposed to using the stand-alone single user file system.

The build scripts described above will create the global array libraries so that global array accesses will be made through the mumps server (`mumpsd`).

To start the server, change directory to `/etc/mumps` and start the server, as root, with the command

```
./mumpsd > log &
```

To halt the server, send it a `SIGINT` (\^C) such as:\n
```
kill -2 1234
```

where 1234 is the `mumpsd` process id. If you do not start the Mumps server, global array access will be unavailable.

If you do a proper Mumps demon shutdown, the database will be intact. However, failure to properly close the server could lead to catastrophic data loss.

The server communicates with the clients by means of sockets.

The Mumps global array data base (`key.dat` and `data.dat`) will be in `/etc/mumps`.

---

3 Note: during normal Linux shutdown, this signal is automatically sent to each process so `mumpsd` will be properly shutdown.
On some multi-core systems, a slight performance improvement may be gained by attaching the Mumps demon to one CPU. To do so, use the following command as root:

```
schedtool -a 0x1 PID
```

where PID is the process id of the demon and 0x1 means the first (cpu0) processor core (0x2 means cpu1 etc.).

1.8.1.2 Native B-tree Database (Standalone)

nativeConfigure.script
BuildMumpsWithGlobalsInNative.script

Builds two stand-alone versions of the mumps interpreter both of which use the native B-tree global array file system, the fastest global array database option.

The first script, nativeConfigure.script, installs as needed any missing system software and then invokes the second script.

The second script, BuildMumpsWithGlobalsInNative.script, assumes that required system software has been installed. It compiles and installs Mumps.

When using a native btree global array database, the database is stored in two files: key.dat (the B-tree) and data.dat (the stored data). Normally these reside in the same system directory as the executing Mumps program.

A given system may have multiple global array databases in multiple directories but each database is completely separate from each other.

Because many applications tend to write/update the database infrequently but read it frequently, one version of the Mumps interpreter is read-only while the other is read-write with respect to a global array database.

The read-only version of Mumps may, as the name suggests, only read elements of a global array database while the read-write version may both read and write elements.

The read-write executable is named mumps while the read-only executable is named mumpsRO. Both are placed in /usr/bin.

In standalone mode, multiple instances of Mumps programs operating on the same database files (key.dat and data.dat) cooperatively share the database. Each instance is given a slice of transactions on the database before it must relinquish control. This can be set by configure (see page 22).

1.8.2 Relational Database Server Resident Global Arrays

1.8.2.1 Overview

The Mumps global arrays my be stored in a relational database system. The two currently supported are MySQL (Oracle Corporation) and PostgreSQL (PostgreSQL Global Development Group). With simple code changes, other servers could also be accommodated.

There are advantages and disadvantages to storing globals arrays in a relational database. The hierarchical nature of the Mumps database is ordinarily not well suited to the tabular structure of a relational database and access is slower.

On the other hand, relational databases provide flexible multi-user, robust, fully ACID (Atomicity, Consistency, Isolation, Durability) compliant data storage along with a complete suite of transaction processing functions not otherwise available in the Mumps language definition.

A further advantage is that global array data may be interrogated and manipulated by ordinary, standard SQL commands.
By default, the Mumps interpreter maps global array references to a multi-column relational database table with the same name as the global array. The columns of the table are named \( a_1, a_2, \ldots, a_{10} \) and so forth. The values in the columns are the indices from a global array reference.

The final column contains the value stored at the reference, if any. For example, the code:

```mumps
set `birds(1,2,3,4,5)="ducks"
```

would map to a table named `birds` in the relational database as follows:

```
birds
+--------+--------+--------+--------+--------+--------+
| a1     | a2     | a3     | a4     | a5     | a6     |
+--------+--------+--------+--------+--------+--------+
| 1      | 2      | 3      | 4      | 5      | ducks  |
```

The total number of columns for a global array is set either to a default number (a `configure` option) or by the `sql/f` Mumps command. The `sql/f` command for the above would look like:

```mumps
sql/f birds 6
```

where the first operand is the table name to be created and the second is the number of columns (including the final column for data values).

If you do not predefine a table, a default number of columns will be used (currently 11).

If your program instantiates array elements like the following:

```mumps
set `birds(1)="all"
set `birds(1,2)="flying"
set `birds(1,2,3)="water"
set `birds(1,2,3,4)="large"
set `birds(1,2,3,4,5)="ducks"
set `birds(1,3)="flightless"
set `birds(1,3,3)="water"
set `birds(1,3,3,4)="large"
set `birds(1,3,3,4,5)="penguins"
```

The relational table will look like:

```
birds
+--------+--------+--------+--------+--------+--------+
| a1     | a2     | a3     | a4     | a5     | a6     |
+--------+--------+--------+--------+--------+--------+
| null   | null   | null   | null   | all    |        |
+--------+--------+--------+--------+--------+--------+
| null   | null   | null   | null   | flying |
+--------+--------+--------+--------+--------+--------+
| null   | null   | null   | null   | water  |
+--------+--------+--------+--------+--------+--------+
| null   | null   | null   | null   | large  |
+--------+--------+--------+--------+--------+--------+
| null   | null   | null   | null   | ducks  |
+--------+--------+--------+--------+--------+--------+
| null   | null   | null   | null   | flightless |
+--------+--------+--------+--------+---------+
| null   | null   | null   | null   | water  |
+--------+--------+--------+--------+---------+
| null   | null   | null   | null   | large  |
```

4 By default, the columns `varchar` (note: the character length is a settable option but the index columns are normally `varchar(64)` while the data column, the last column, is normally `varchar(512)`). The character size of columns can be set to other values by `configure`. Smaller values may improve performance.

5 Table row order may differ but this is not important.
Mumps access requests produce the expected results:

| write $^birds(1) => all |
| write $^birds(1,2) => flying |
| write $^birds(1,2,3) => water |
| write $^birds(1,2,3,4) => large |
| write $^birds(1,2,3,4,5) => ducks |

The row-wise duplication in the above is also present in many other Mumps systems and the nulls have little effect on overall performance. Ideally, however, a more dense table is easier to deal with from a SQL frame of reference.

An advantage, as mentioned above, is that data stored in such a table may be queried by an ordinary SQL command such as:

```sql
select a6 from birds where a1='1' and a2='2' and a3='' and a4='' and a5='';
```

which yields flying. Note the specific use of null strings.

Similarly, SQL views may be established on the birds table to facilitate access.

### 1.8.2.2 Basic Database Configuration

By default, in order for Mumps to store and retrieve global arrays from a relational server the following requirements must be met:

1. There must be a database user named mumps authorized to create, drop, read and write tables;
2. There must be a database named mumps with privileges sufficient to create, destroy, read and write tables.

The user name mumps and the database name mumps may be changed in the configure procedure. By default, user mumps has the default password abc123 which may also be changed with configure.

Note: if you want to experiment with this before committing it to your main host, you might try building a virtual Linux machine with Linux Mint and Oracle’s Virtual Box (both are free).

### 1.8.2.3 Relational Database Configure Options Common to All Servers

The following configure options are common to all relational database clients. You probably do not want to change these.

#### 1.8.2.3.1 --with-datasize=numeric-value

The maximum length of a string stored at a global array node in the last column. Performance is improved if this value is as small as possible. If an element stored at a global array node exceeds this length, it will be truncated. Default: 512.

#### 1.8.2.3.2 --with-dbname=name

The name of the database in the relational database server where Mumps will store the global arrays. Default: mumps.

#### 1.8.2.3.3 --with-indexsize=numeric-value

Specify the size of the varchar declaration of columns a1, a2, ... This is the maximum string length of any individual global array index element. For example, if your global array reference is: ^a(1,2,3), you have three index columns, (a1, a2, and a3) and a data columns (a4). This option sets
the maximum string length of columns a1, a2, and a3. Performance is improved if this value is as small as possible. If an individual element of a global array index exceeds this length, it will be truncated. Default: 64.

1.8.2.3.4 --with-tabsize=number

Maximum number of index elements in a global array reference not counting the name of the global array itself. Default: 10. This is the maximum depth of any global array tree. The maximum permitted value is 31.

1.8.2.4 Initialization of a Mumps Relational Database

The first time you use a table in the relational database, you must initialize it with the Mumps command:

```
sql/f global_name columns
```

where global_name is the name of the global array. When you do this, you may see a warning message that the global_name table does not exist. This can be ignored. A second running of the initialization command will not show the message.

If you do not initialize a global array before using it, a default ten column array will be constructed.

1.8.3 Relational Database Server Connection for Mumps Global Arrays

In order to store the global arrays in a database server, you need to create a user named mumps and a database named mumps (these defaults can be changed by means of configure).

You also need to inform the Mumps client code of the password for the mumps user. This can be set by means of configure or in btree.cpp.in (in the section corresponding to the server (MySQL or PostgreSQL) you are using - see below). The default password for user mumps is abc123.

For PostgreSQL, the connection information, including user, database and password, is found in function AllocSV() in file sysfunc.cpp.in:

```
strcpy(p1->Connection,"host=@remotehost@ dbname=@dbname@ user=@user@
password=@passwd@
"
```

For MySql, this information is found in file btree.cpp.in the code:

```
char host[128]="@host@";
char user[128]="@user@";
char passwd[128]="@passwd@";
char dbname[128]="@dbname@";
unsigned int port=@port@;
char socket[128]="@socket@";
```

Items enclosed in @-signs are replaced by configure. The values refer to connection options for the respective database servers. The defaults are database name: mumps; user: mumps; password: abc123; and host: localhost.

These options can be set by configure.

1.8.4 MySQL Database Option

mysqlConfigure.script
BuildMumpsWithGlobalsInMySQL.script

The script mysqlConfigure.script (1) installs any required system software (including MySQL), configures MySQL, and (2) invokes the second script to compile and build the MySQL client mumps interpreter.

---

6 It is due to an SQL DROP statement on the table relation before table is built.
The MySQL required software is listed in section 1.6 on page 7.

`BuildMumpsWithGlobalsInMysql.script` compiles and builds a MySQL client `mumps` interpreter. It assumes that required system software has already been installed.

The MySQL client Mumps interpreter requires a properly configured and running MySQL server for global array access.

The script `mysqlConfigure.script` is mainly useful as a quick start for an installation which does not already have a MySQL server running on it. If the script detects the directory `/usr/lib/mysql` it assumes that MySQL is present and does not attempt to install MySQL. It proceeds, instead, to determine if any system software is missing and install as necessary. It then invokes the second script.

If you have an existing MySQL server, you may want to manually set up the necessary `mumps` user and password along with any missing system software.

Depending on whether your host machine will be client, server, or both, you will need to install the appropriate MySQL software. This will include the appropriate development libraries. If you receive an error message during compilation of link edit, it is probably due to missing libraries.

### 1.8.4.1 MySQL Manual Installation

The script file `mysqlConfigure.script` can be used to install and configure Mumps and MySQL on a system that does not already have MySQL or Mumps installed. It identifies and installs missing system software, configures Mumps, and initializes the MySQL database.

The script assumes you are working on a version of Linux (such as Mint, Ubuntu, Debian, etc.) that uses the `apt-get` to install and upgrade software. If you are not, you will need to manually install the packages itemized in the script.

If you already have MySQL installed, this script will install any parts that are missing (such as the development libraries) and upgrade others. The script will also install or upgrade other system software needed by Mumps.

If you do not want some or all of the packages upgraded, do not use this script. Manually install the missing software.

If MySQL was not previously installed, you will be prompted to give a password to be used for the MySQL root user. You must do so since you will need this password later in the procedure. The installation will otherwise use standard MySQL defaults.

After the software is installed, the script to configure Mumps is called (`BuildMumpsWithGlobalsInMySQL.script`). If, after the initial MySQL installation, you need to change a configuration setting in Mumps, you may invoke this script directly. Its options are given in section 1.8.4.2 on page 14 below.

### 1.8.4.2 MySQL Installation Options

The following are the `configure` options for MySQL and their default values.

1. `--with-mysql-user=user`
   
The MySQL userid of the client Mumps program to be used when establishing a connection. May be set in the `sql/d` connection string. Default: `mumps`

2. `--with-mysql-host=nbr`
   
The IP number of the MySQL server. May be set in the `sql/d` connection string. Default: `localhost`

3. `--with-mysqldb`
   
   Enables MySQL database storage of globals. MySQL is not enabled unless this option is
specified. May not be set by sql/d connection string. Incompatible with the corresponding PostgreSQL enabling option.

4.  --with-mysql-passwd=val

   Specify, if needed, the MySQL user passwd. May be set in the sql/d connection string. Default: the empty string (no password).

5.  --with-mysql-port=nbr

   Port number to access the MySQL server. Default: 0.

6.  --with-mysql-socket=nbr

   Socket through which to access the MySQL server. Default: NULL.

1.8.4.3 Mumps Build for MySQL Resident Global Arrays

   The script BuildMumpsWithGlobalsInMySQL.script contains the code to build and configure Mumps to use a MySQL server. This script also sets the MySQL option for the compiler and the toolkit.

   The script file mysqlConfigure.script can be used to install and configure MySQL software. It then invokes BuildMumpsWithGlobalsInMySQL.script. After you have installed MySQL, use BuildMumpsWithGlobalsInMySQL.script to make configuration changes.

1.8.4.4 Configuring a Remote MySQL Server

   See MySQL documentation.

1.8.4.5 Using MySQL Resident Global Arrays from Mumps

   Same as for PostgreSQL with the exception of differences in the connection string used by sql/d.

1.8.4.6 Command Line Interpreter mysql

   The MySQL command line interpreter is named mysql. You may invoke it for user mumps and password abc123 with:

   mysql -u mumps -pabc123

   To use the mumps database, type:

   use mumps;

   Other options are:

   1.  show tables; Displays the tables in the database.
   2.  show columns from abc; Displays the columns from table abc.
   3.  set global innodb_flush_log_at_trx_commit=0; Improves transaction speed at the expense of reliability (see MySQL documentation).

   Note: in MySQL you may not use some table names. One of these is index.

1.8.5 PostgreSQL Database Option

   postgressConfigure.script
   BuildMumpsWithGlobalsInPostgreSQL.script

   The script postgressConfigure.script installs required system software (including PostgreSQL), configures PostgreSQL, and then invokes the second script to compile and build a PostgreSQL client mumps interpreter.

   The second script, BuildMumpsWithGlobalsInPostgreSQL.script, builds a Mumps PostgreSQL client. The PostgreSQL client mumps interpreter requires a properly configured and running PostgreSQL
server if access is made to a global array. After you have installed PostgreSQL, use
BuildMumpsWithGlobalsInPostgreSQL.script to make any configuration changes.

The required PostgreSQL software is listed in section 1.5 on page 7.

In general, overall performance appears to be better if you use PostgreSQL rather than MySQL.

### 1.8.5.1 PostgreSQL Options

Mumps permits storage of global arrays in PostgreSQL database tables. Using PostgreSQL gives
the Mumps user a fully ACID (Atomicity, Consistency, Isolation, Durability) compliant database but
database access will be slower overall.

When you create/store global arrays, they will be stored, by default, in a database known as
mumps on the PostgreSQL server. The tables created in this database will have the same names as
the corresponding Mumps global arrays and may also be accessed from non-Mumps clients by
means of SQL SELECT and related statements.

When using a PostgreSQL server, it is possible to construct views of database tables so that they
can be directly accessed by Mumps as global arrays. An example of this is given below.

When storing global arrays on a PostgreSQL server it may be desirable, when beginning a series
of related global create/store/update transactions, to precede the Mumps code with:

```
SQL BEGIN TRANSACTION;
```

ultimately to be followed by:

```
SQL COMMIT;
```

This permits the Mumps global array create/store/updates to run faster. It also insures that the all
the transactions will run without interference from other users. This eliminates the need for the
Mumps lock command.

However, should there be a failure before the final COMMIT, the uncommitted data may be lost.

### 1.8.5.2 Command Line Interpreter psql

Assuming the default database configuration (see Quick PostgreSQL Installation in section
1.8.5.3.1), you may access the PostgreSQL command line interpreter with:

```
psql mumps
```

From this program, you may access the Mumps tables. Also note that there is a modified version
of psql that permits execution of Mumps commands from the CLI.

### 1.8.5.3 Installing PostgreSQL

Note: as of this writing, the PostgreSQL release is 9.3 and this number is used in the following
documentation. Check which version you have and adjust the following accordingly. Subsequent
versions of PostgreSQL may have different interfaces and may use different libraries which may
invalidate some or all of the following.

#### 1.8.5.3.1 Quick PostgreSQL Installation

If your system is Ubuntu-based and you do not presently have PostgreSQL installed or you have a
more or less standard PostgreSQL installation, there is a script file in the distribution that should be
able to build and install both Mumps and PostgreSQL. The script is named:

```
postgressConfigure.script
```

You will want to edit this script to tailor it to your needs. Instructions are contained in the
comments. This script must be run as root. It was developed using Linux Mint 17.2 and PostgreSQL
version 9.3 and should work with related distributions with a change of version number as needed.
1.8.5.4 PostgreSQL Configuration

You may want to modify some PostgreSQL configuration options. These configuration options are usually found in:

```
/etc/postgresql/9.3/main/postgresql.conf
```

Note: replace 9.3 but the current version of your installation.

For example, in order to suppress extraneous notices (as opposed to warning and error messages) from appearing in the output of a PostgreSQL client (e.g., Mumps), you may want to set the following configuration parameter:

```
client_min_messages = warning
```

1.8.5.5 PostgreSQL Specific Mumps Install Options

The following detailed setup instructions apply to Ubuntu and Ubuntu-like distros such as Mint. In Red Hat based distros, some PostgreSQL files may be located in different directories and this may affect the installation procedures. The instructions here are based on Linux Mint Mate 17.

If you have not installed PostgreSQL or only have a basic PostgreSQL installation, see section 1.8.5.3.1 above which does most of the following automatically.

The primary purpose of the following is to create a database named `mumps`, a PostgreSQL user named `mumps` with an initial password of `abc123`. The Mumps PostgreSQL, by default, logs into the server as `mumps` and creates it's relational tables in the `mumps` database.

1. Using your package manager (e.g., Synaptic) install the latest version of PostgreSQL including the development libraries. See section 1.5 on page 7.

2. Configure PostgreSQL options are:

```
configure prefix=/usr \
--with-pgdb=/usr/include/postgresql \
--with-dbname=mumps \
--with-pgsql-host=127.0.0.1 \
--with-pgsql-user=mumps \
--with-pgsql-passwd=abc123
make
make install
```

Note: different versions of PostgreSQL have had a habit of playing hide the files which may cause problems. Check for updates if you experience problems. The line:

```
--with-pgdb=/usr/include/postgresql
```

Tells configure where the include files are located. This is currently the default location.

The other options indicate:

1. `prefix`: where to place the Mumps executables.
2. `--with-dbname`: the name of the database in which to store the global arrays.
3. `--with-pgsql-host`: the IP number of the machine hosting the server (127.0.0.1 is localhost)
4. `--with-pgsql-user`: the user name that Mumps will login to the server as.
5. `--with-pgsql-passwd`: the password to be used by Mumps when logging in.

The server must be running and properly configured in order for the Mumps global array facility to function.7

---

7 Note: it appears that some Synaptic package manager installs may incorrectly address the location of the Postgresql socket. If, upon starting mumps, you get a message that the connection could not be opened and to check the socket, you will need to correct an entry in the file:
Once you have built a Mumps database in PostgreSQL, you may query it with general purpose SQL commands (such as SELECT, discussed elsewhere).

1.8.5.5.1 Configuring the listen_address

By default, Mumps logs into the PostgreSQL server on the current machine. If you want to run Mumps programs on a different machine than the one running Mumps, you need to enable, on the server machine, connections to its PostgreSQL server.

The following are some brief instructions on how to permit a remote server to process Mumps requests. You should consult the PostgreSQL manuals for details which would be more appropriate to your application:

To accept connections, you should set, on the host machine, as root, the listen_address option in the file:

/etc/postgresql/9.3/main/postgresql.conf

to contain the IP numbers of the systems from which you are willing to accept connections. Note: the intermediate directory 9.3 in the above refers to the current PostgreSQL release number. This will change with time.

For example:

listen_addresses = 'localhost,*'

The above, note the quotes, permits connections from all remote addresses. After altering this setting a restart is required:

/etc/init.d/postgresql restart

The connecting clients' IP numbers should identified be in the file pg_hba.conf found in the same directory. To enable a network connection, you should insert a line into this file. If you are using IPV4 addresses, it should look something like:

host    all      all      10.42.0.0/16      trust

which means that the the high order 16 bits of the IPV4 of the incoming request IP number must match 10.42 but the remaining 16 bits can be any value. The server will accept connections from any machine with the 10.42 prefix. The trust option means that a password will not be required from the connecting client. If you want the user to supply a password, use:

host    all      all      10.42.0.0/16      md5

There are other security options. Consult the PostgreSQL documentation or your system administrator.

You can test the connection to the server from a remote machine with the command:

psql -h 10.42.0.26 -d mumps

where the '-h' option specifies the remote host to connect to and the -d mumps specifies the name of the database. This command assumes that the trust option was used and that the login_userid of the

/etc/postgresql/9.3/main/postgresql.conf

In this file, change the value for the entry unix_socket_directory to point to the directory in the error message (probably: /var/run/postgresql). This is a PostgreSQL issue, not a Mumps issue. You will need to restart the database after this or any other configuration changes:

/etc/init.d/postgresql restart
user on the remote (client) system is the same as an authorized user on the remote (server) system. The server machine is at address 10.42.0.26. To exit from psql, type \q (backslash-q).

If you attempt to use Mumps with PostgreSQL as the database from an account not recognized by PostgreSQL, you will receive the error messages of the form:

```plaintext
*** Connection to database server failed in or near line 0
Error msg: FATAL: role "root" does not exist
Connection string=dbname=mumps
```

The PostgreSQL server can be started with SSL enabled by setting the parameter `ssl` to `on` in `postgresql.conf`.

### 1.8.5.5.2 Performance Tuning

By default, PostgreSQL is set for stringent data protection. This results in considerable disk activity to insure that data is never lost. However, many of these procedures slow the operation of the database during update to a considerable extent. They can, in many cases, be dispensed with with only minimal effect on database integrity.

The main configuration file is `postgresql.conf`. For a server started by the operating system, this file will, by default, be in `/etc/postgresql/9.3/main`.

Alter the settings as follows:

```plaintext
wal_level = minimal
fsync = off
synchronous_commit = off
full_page_writes = off
archive_mode = off
```

The result may be a considerable improvement in speed.

In cases where speed is important, a series of inserts into the database that are done as one transaction is faster than individual transactions (the default). For example:

```plaintext
#!/usr/bin/mumps
sql/f a 4
set k=0
sql SET LOCAL synchronous_commit TO OFF;
sql begin transaction;
for i=1:1:100 do
  s k=k+1
  set ^a(i)=k
for j=1:1:100 do
  .. set k=k+1
  .. set ^a(i,j)=k
for m=1:1:10 do
  ... s k=k+1
  ... s ^a(i,j,m)=k
sql commit
halt
```

One line 3 the table `a` is cleared and initialized to four columns. On line 7 the SQL command disables the PostgreSQL server's from waiting for the transaction's records to be flushed to permanent storage before returning a success indication to the client. This causes the inserts to proceed very much faster but at some risk of data loss (but not data corruption) should the system fail during updates.
Line 8 initiates a transaction and line 19 commits the transaction that finalizes the values stored in the intervening lines.

### 1.8.5.5.3 PostgreSQL Server Connections

Each time you start Mumps you must establish a connection with a PostgreSQL server. The command to do this is `sql/d` but this is normally done automatically the first time a Mumps program references a global array. The default connection settings are normally used.

The `sql/d` command is only used to connect with a server other than the default or to reconnect with the default server if the original connection was terminated.

Normally, a default connection is made automatically when anytime you attempt to use a global array. The defaults are established by `configure`.

The `sql/d` command should only be used when you are making a non-default connection.

The prototype for the connection command sent by a Mumps program (found in the code module `sysfunc.cpp.in`) to the server is:

The `sql/d` command is used when you want to make a connection using connection parameters other than those established as defaults. Normally, you will not use this command.

`Sql/d` may be entered in interactive mode or as part of a Mumps script file. If you want parameters other than the defaults, you must place them on the `sql/d` line and execute the line before any attempt to access the global arrays. Once you attempt to access a global array, the default connection is automatically established.

The options available in `sql/d` and their keywords are the same is given in the PostgreSQL documentation for the `PQconnectdb()` function.

The `sql/d` command causes any current database connection to be closed. The remainder of the line consists of the new connection arguments, some of which may be of the `&~exp~` format. If successful, `$t` will be true. This connection information will be used until changed or the Mumps client terminates. Examples:

```plaintext
sql/d dbname=mumps
sql/d host=abc.def.xyz.edu dbname=mumps
sql/d hostaddr=123.321.432.321 dbname=mumps
sql/d user=joe password=abc123 dbname=mumps host=abc.def.xyz.edu
```

### 1.9 PostgreSQL Transaction Limit

The number of pending transactions between SQL Begin and SQL Commit commands is limited by PostgreSQL. If this number (implementation defined) is exceeded, transactions will be lost. Check the PostgreSQL log in `/var/log/postgresql`.

### 1.10 Use with Apache

When running through the CGI-BIN interface with the Apache web server, be sure your files and directories are not owned by `root`. Make them owned by Apache (user `www-data` in Linux Mint 13) and in the Apache group (also `www-data`). Apache’s default cgi-bin directory is `/usr/lib/cgi-bin`. You will need to be `root` to add/modify files in this directory. Be sure to make Apache (`www-data`) an authorized PostgreSQL user with the `createuser` command (see 1.8.5.5 above).

### 1.11 Use with Windows

If you install all the relevant Cygwin software, you may build a native database version of Mumps in Windows using Cygwin for the native and relational database versions. An executable (`mumps.exe`) so built, and may be copied to another directory and executed from a normal Windows

---

8 See section 5.14 on page 39.
command prompt if you copy several Cygwin DLLs to your system (\Windows) directory. These are:

cygcrypto-1.0.0.dll
cygpcre-1.dll
cygstdc++-6.dll
cygz.dll
cygmysqlclient-18.dll
cygssl-1.0.0.dll
cygwin1.dll
cyggcc_s-1.dll

The first group is needed for all versions while the second group is also needed for MySSQL versions. These can be found in the Cygwin /bin directory.

The cygmysqlclient-18.dll is only needed for the MySQL build. (Note: version numbers may be different when you read this).

If you build a MySQL version in Cygwin, even though it is running under Cygwin, Mumps will access the MySQL server running on Windows (or at a remote machine if you specify a remote IP number). You may run Mumps natively in a Windows command prompt box without Cygwin if you install the DLLs noted above in your Windows folder.

1.11.1 Windows/Cygwin Install

The installation scripts Build... will work with Cygwin if the appropriate software is installed. However, the configuration scripts nativeClientServerConfigure.script, nativeConfigure.script, postgresConfigure.script and mysqlConfigure.script will not work.

1.12 Math Options

Arithmetic in this Mumps distribution can be performed either by hardware or by a library of extended precision software.

In extended precision mode, the precision of both floating point and integer numbers can be significantly larger than is the case with standard hardware arithmetic with minimal performance penalty.

The several Build scripts look for files gmp.h and mpfr.h. If these are found, they cause the build to use the extended math packages. If not, the builds will use hardware arithmetic.

You may override this and force hardware arithmetic by modifying the scripts to add the --with-hardware-math option.

1.13 Numeric Configuration Options

Both extended precision and basic hardware precision are available as noted above.

In hardware precision mode, floating point numbers are processed by the machine’s arithmetic processing hardware. Floating point numbers are treated as 64-bit double values and integers are treated as signed 64-bit long integer values.

Hardware integers range from -9223372036854775807 (-2^63 +1) to 9223372036854775807 (2^63 -1).

Hardware floating point numbers utilize a one bit sign, an 11 bit exponent and a 52 bit fraction. This translates into approximately 16 decimal digits of precision in the range of ± ~10^{321.3} to ~10^{308.3}.

9 Only limited testing is done on Windows. If you want to run Mumps on Windows, use a virtual machine package such as Oracle’s Virtual Box and install Linux Mint Mate.

10 Note: version numbers may change with time.
Extended precision is available through use of the GNU multiple precision arithmetic library\(^{11}\) and the GNU MPFR library\(^{12}\). For integers, this means effectively unlimited precision. For floating point numbers, the exponent is 64 bits and the fraction is user specified (default of 72 bits in Mumps - this option may be set by configure).

Hardware arithmetic will be selected during compilation of the interpreter if (1) configure does not find the extended precision libraries or the user specifies the configuration option:

```
--with-hardware-math.
```

If extended precision is used, the number of bits in the fraction of a floating point number can be set with:

```
--with-float-bits=value
```

where value is the number of bits. The default value is 72.

For extended precision floating point numbers, the number of digits of precision to print is controlled by:

```
--with-float-digits=value
```

where value is the number of digits. The default is 20.

When printing a number, the number of digits being printed should be consistent with the number of bits in the fraction. If the number of digits to print is too large, random low-order digits may appear in numbers.

### 1.14 All Configure Options

The basic install sequence, as is the case with many Linux based packages is to run something similar to the following as root:

```
./configure prefix=/usr
./make
./make install
```

The configure step, however, as is typical, contains many options. Specifying these causes modification to the source code and changes the final product.

The distribution, as noted above, contains several bash script files with pre-configured configure commands. For the most part, you probably don't want to write your own configure options except in limited cases. You may, however, want to edit the files provided to set details such as passwords and so on. This is discussed below.

The full set of options to configure are:

1. configure prefix=/usr

The directory where the runtime modules will be stored. If this is not specified, the default location is in a directory named mumps_compiler in the user's home directory. Normally, if you want Mumps available to all users, you will specify the option as shown and run make and make install as root. If you specify /usr as shown, the Mumps routines will be placed in /usr/bin/mumps.

2. PostgreSQL options

   a) --with-pgsql-user=userid
   b) --with-pgsql-passwd=password
   c) --with-pgsql-host=IP

   userid for PostgreSQL server account [mumps]
   password to access database [“abc123”]
   select host IP number [127.0.0.1]

---

d) --with-pgdb= location of libraries [/usr/include/postgresql]

3. MySQL options

a) --with-mysql-user=userid userid for MySQL server account [mumps]
b) --with-mysql-host=IP IP number of remote host [127.0.0.1]
c) --with-mysqldb Enable MySQL data base for globals
d) --with-mysql-passwd=val Select mysql user passwd ["abc123"]
e) --with-mysql-port=nbr Select mysql port ["]
f) --with-mysql-socket=nbr Select mysql socket ["]

4. General Relational Database Options

a) --with-dbname=name SQL data base name [mumps]
b) --with-indexsize=number SQL DB index max [64]
c) --with-tabsize=nbr number of columns in SQL table mumps [10]

5. Native Database Options

a) --with-slice=value The number of database transactions an instance of a standalone native B-tree Mumps programs may perform on the database before relinquishing control. Default: 10

b) --with-cache=VAL native globals cache size [65537]

The only legal values for this parameter are:

9
17
33
65
129
257
513
1025
2049
4097
8193
16385
32769
65537
131073
262145
524289
1048577

c) --with-block=blksize native btree block size [8192]

The native Btree database consists of two files: the tree file (keydat) containing the actual Btree and the data file (data.dat) containing stored data. The maximum size of the Btree file is dependent on the block size. The block sizes listed below each have a PAGE_SHIFT value and this ultimately determines the maximum file size as shown. The basic internal disk address is effectively 31 bits (signed 32 bit quantity) but, depending upon the block size, some number of bits at the low-order end are always zero. For example, if the block size is 1024, the final 10 bits of an address are always zeros. As only the significant 31 bits are stored, the true address is not 31 bits but 41 bits thus a file size of 2 terabytes is possible.

The only legal values for this parameter are:

1024
2048
The block size determines the internal PAGE_SHIFT factor:

- 1024 → PAGE_SHIFT 10
- 2048 → PAGE_SHIFT 11
- 4096 → PAGE_SHIFT 12
- 8192 → PAGE_SHIFT 13
- 16384 → PAGE_SHIFT 14
- 32768 → PAGE_SHIFT 15
- 65536 → PAGE_SHIFT 16
- 131072 → PAGE_SHIFT 17
- 262144 → PAGE_SHIFT 18
- 524288 → PAGE_SHIFT 19
- 1048576 → PAGE_SHIFT 20
- 2097152 → PAGE_SHIFT 21

PAGE_SHIFT 10 corresponds to MBLOCK 1024 and a max Btree file size of 2 TB
PAGE_SHIFT 11 corresponds to MBLOCK 2048 and a max Btree file size of 4 TB
PAGE_SHIFT 12 corresponds to MBLOCK 4096 and a max Btree file size of 8 TB
PAGE_SHIFT 13 corresponds to MBLOCK 8192 and a max Btree file size of 16 TB
PAGE_SHIFT 14 corresponds to MBLOCK 16384 and a max Btree file size of 32 TB
PAGE_SHIFT 15 corresponds to MBLOCK 32768 and a max Btree file size of 64 TB
PAGE_SHIFT 16 corresponds to MBLOCK 65536 and a max Btree file size of 128 TB

The data file may grow to a max of 2**64 bytes for all settings.

d) --with-client build native btree client data base code
e) --with-server-dir native Btree server home directory [/etc/mumps]
f) --with-readonly native database will be readonly – only applied to the native
global array facility

6. --with-ibuf= max size interpreted program [32000]
7. --with-strmax= max internal string size [4096]
8. --with-locale=locale locale information [en_US.UTF-8]
9. --with-terminate-on-error halt interpreter on error [off]
10. --with-includes=DIR to identify header dirs (Apple build only)
11. --with-libraries=DIR to identify libs (Apple build only)
12. --with-float-bits=val number of bits in floating point fractional part (72)
13. --with-float-digits=val number of decimal digits to print in a floating point number (20)
14. --with-hardware-math use hardware arithmetic facilities
2 Running a Mumps Program

2.1 Start the Global Array Server

Note: if you are using either the PosgreSQL, MySQL or native client /server option to store global arrays, you must start the PostgreSQL or MySQL server prior to attempting to access any global array.

2.2 Mumps CLI Interpreter

To run the command line interpreter from a terminal window, type:

mumps

Any Mumps commands you enter will be executed immediately. To exit the interpreter, type H[alt].

In interactive mode, you will be presented with a prompt (>). Any Mumps command may be typed for immediate execution (including a goto or do commands with a file name reference pointing to a file to be loaded and executed).

The keyboard up arrow and down arrow keys may be used to cycle through and display commands previously entered during this session.

A previously entered command my be re-executed by using the keyboard up arrow key to locate and display the command and then typing <enter>.

Input to the Mumps CLI follows GNU readline conventions.

2.2.1 Mumps CLI Special Commands

2.2.1.1 \globals

Lists the names of the global array tables and the number of columns in each (works only when using PostgreSQL or MySQL).

2.2.1.2 \halt \quit \h \q

Exit the Mumps CLI. The Mumps Halt command works as well.

2.2.1.3 \sys cmd

Executes cmd in a system shell then returns to the Mumps CLI. Example:

\sys ls -lh *

Expression substitution is permitted\textsuperscript{13}:

> write $zsqlOutput
9910.tmp

> sql select * from doc where a1='1' limit 10;
> \sys cat &~$zsqlOutput~

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>acetaldehyde</td>
<td>ribonuclease</td>
<td>alteration</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\textsuperscript{13} Note: The default contents of $zsqlOutput are the process id of the Mumps program followed by the .tmp extension.
2.2.1.4 \texttt{\textbackslash mumps\ pgm}

Runs the code in \texttt{pgm} as a Mumps program. The code need not begin with the \texttt{bash} interpreter comment. Expression substitution is permitted:

\begin{verbatim}
> \sys cat xxx.mps
  for i=%1:%2:%3 write i," \\
  write !
> \mumps xxx.mps 1 2 10
  1 3 5 7 9
> set x="1 2 10"
> set y="xxx.mps"
> \mumps &~y~ &~x~
  1 3 5 7 9
\end{verbatim}

2.2.1.5 \texttt{\textbackslash sql sqlCommand}

Send \texttt{sqlCommand} to the PostgreSQL server and display any table output returned. Example:

\begin{verbatim}
> \sql select * from doc where a1='1' limit 10;
\end{verbatim}

\begin{verbatim}
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>acetaldehyde</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>ribonuclease</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>alteration</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>catalytic</td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td>phosphate</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>ribonuclease</td>
<td>11</td>
</tr>
<tr>
<td>1</td>
<td>react</td>
<td>12</td>
</tr>
<tr>
<td>1</td>
<td>acetaldehyde</td>
<td>13</td>
</tr>
<tr>
<td>1</td>
<td>cyanoborohydride</td>
<td>15</td>
</tr>
</tbody>
</table>
\end{verbatim}

Expression substitution is also possible:

\begin{verbatim}
> set i="doc",j=10
> \sql select * from &~i~ where a1='1' limit &~j~;
\end{verbatim}

\begin{verbatim}
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>acetaldehyde</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>ribonuclease</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>alteration</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>catalytic</td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td>phosphate</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>ribonuclease</td>
<td>11</td>
</tr>
<tr>
<td>1</td>
<td>react</td>
<td>12</td>
</tr>
<tr>
<td>1</td>
<td>acetaldehyde</td>
<td>13</td>
</tr>
<tr>
<td>1</td>
<td>cyanoborohydride</td>
<td>15</td>
</tr>
</tbody>
</table>
\end{verbatim}

2.3 Mumps Programs (scripts)

Mumps programs are ASCII files that can be created by any ASCII text editor. Do not use word processing editors that may embed hidden formatting characters into the text.

A script will normally have the following as their first line:
#!/usr/bin/mumps

The file extension of a Mumps program .mps is preferred but not required.

The Mumps source file must be made executable:

    chmod u+x prog.mps

where prog.mps is the name of your mumps source file.

Example:

    #!/usr/bin/mumps
    for i=1:1:10 do
        . write "Hello World ",i,!
    halt

You may execute the program by typing prog.mps to your terminal prompt. The program above will write Hello World, followed by a number ten times.
3 Relational Database Commands & Variables

3.1 Creating Global Array Relational Database Tables

As discussed above, you may enable storage of global arrays in either the MySQL or PostgreSQL relational database systems. While access to globals is slower than is the case with the native global array handler, the reliability and transaction processing capabilities offer many advantages.

In either MySQL or PostgreSQL, the globals will be stored in database tables that have the same names as the corresponding global arrays.

You should create the database tables before attempting to store globals in them or a default declaration will be done automatically. The default declaration will assume a predetermined number of columns (a configure option) which may not be correct.

Global arrays should be pre-declared with the sql/f command (see section 1.8.2.4 on page 13).

When a global array is stored in a table, the names of its columns are a1, a2, a3, ... The number of columns is set when the table is created, ordinarily by sql/f. However, you may use the SQL command ALTER TABLE to add or drop a column14.

The built-in Mumps pseudo-variable $zTabSize (spelling is case insensitive) may be used to determine the number of columns in the most recently accessed table. $zTabSize is updated each time a global array is accessed.

The current table name in use is found in the system variable $zTable (spelling is case insensitive). This is the name of the most recently accessed global array table.

Neither $zTabSize nor $zTable are valid in a program until you have made at least one global array access.

Tables created by Mumps may be accessed outside of Mumps with standard SQL commands. The table name is the same as the global array name and the columns are a1, a2, a3, ... etc. The data stored at a global array reference is always the last column.

3.2 Mumps Access to Relational Tables Not Created by Mumps

Tables or views created by programs other than Mumps may be accessed by Mumps if their column names and data types conform to Mumps or an appropriate VIEW for each is created that maps their actual column names to the column names used by Mumps (a1, a2, a3, ...). Mumps expects that the contents of each column be a text or character data type in order for all Mumps global array access functions to work correctly.

In your Mumps program, the non-Mumps table will be visible as a global array with the same name as the table. Note, however, the last column of the table will be interpreted as the value stored for the global array reference.

For example, the SQL commands in Figure 1 create and populate a small RDBMS table of temperature and dew point named temps with column names city, temp, and dewpt.

In order to make this table visible to Mumps, we map it to a view named mtemps that renames the columns to be compatible with Mumps. Note that the final column, a4, where Mumps normally stores values stored for a global array reference defined by the row, is defined as a constant empty string for each tuple. Also note that the data types of the columns are all a form of varchar.

drop view if exists mtemps;
drop table if exists temps;
create table temps ( city varchar(32), temp varchar(10), dewpt varchar(10));

14 If you use the ALTER TABLE command form a Mumps program, you should exit/restart the Mumps program before attempting to access the altered table so that Mumps may update its internal tables.
insert into temps values ('Boston', '32', '25');
insert into temps values ('Hyannis', '42', '32');
insert into temps values ('Norwood', '32', '12');
insert into temps values ('Quincy', '32', '24');
insert into temps values ('Waltham', '28', '23');

create view mtemps (a1,a2,a3,a4) as
    select city as a1, temp as a2, dewpt as a3, text '' as a4
    from temps;

select * from mtemps;

Figure 1 Creating a View

Figure 2 gives a mumps program to access the view from Figure 1.

#!/usr/bin/mumps

    write "mtemps",!

    for city=$order(^mtemps(city)) do
        . for temp=$order(^mtemps(city,temp)) do
            .. for dewpt=$order(^mtemps(city,temp,dewpt)) do
                ... write city,?15,temp,?20,dewpt,!

mtemps
Boston        32  25
Hyannis       42  32
Norwood       32  12
Quincy        32  24
Waltham       28  23

Figure 2 Mumps View Access

Alternatively, if the underlying table from which the view will be created contains numeric quantities, these can be cast as character strings in the view as shown in Figure 3.

drop view if exists mtemps;
drop table if exists temps;

create table temps ( city varchar(32), temp int, dewpt int);

insert into temps values ('Boston', '32', '25');
insert into temps values ('Hyannis', '42', '32');
insert into temps values ('Norwood', '32', '12');
insert into temps values ('Quincy', '32', '24');
insert into temps values ('Waltham', '28', '23');

create view mtemps (a1,a2,a3,a4) as
    select city as a1, to_char(temp, '999') as a2,
        to_char(dewpt,'999') as a3, text '' as a4
    from temps;

select * from mtemps;

Figure 3 View with Data Conversion

Using a similar technique, most RDBMS tables can be viewed in Mumps. Note: MySQL and PostgreSQL at this time do not permit alteration of view values so the tables are read-only. Also note, the example in Figure 3 uses PostgreSQL data conversion functions. MySQL uses different functions.
3.3 SQL Commands in Mumps

If relational database storage of globals is enabled, the following commands are available in the Mumps interpreter. If the native database is in use, these are ignored.

3.3.1 sql string

The remainder of the line is passed to the SQL server. The line should contain a valid SQL command which is normally terminated by a semi-colon.

Any text of the form &~exp~ will result in exp being evaluated with the result replacing &~exp~.

Mumps does not check the validity of the SQL command. The builtin Mumps variable $zsql will contain any messages returned by the RDBMS server or 'ok' if there were none.

3.3.2 sql/c SQL Disconnect

The sql/c command disconnects from the SQL server. This is normally done automatically when a program terminates. No other text may appear on this line.

3.3.3 sql/f Format SQL Table

The form: sql/f instructs the relational database server to create and initialize a table to be used to store global arrays and to delete any previous contents the table may have had.

The sql/f command has two arguments:

1. a relational table name and
2. the number of columns in the table, including the column to store data.

The arguments must not contain embedded blanks. They are separated from one another by blanks.

The first argument is the name of the relational table to be created. It's name must conform to the naming requirements for a Mumps variable (no underscore characters, for example). It should be lower case as some relational database systems do not differentiate between upper and lower case table names.

The numeric argument may range between 1 and 20. It is the number of columns in the database that will contain stored data. This number is the maximum number of global array indices, minus one, for any global array reference to be stored in this table (the highest numbered column is for any stored values). Thus, if you will not have more than 10 indices in a table, this number should be 11. The first 10 will be used for global array indices and the 11th for any stored data. The minimum value permitted is 2.

Examples:

     sql/f labs 3
     sql/f meds 5

Variables or expressions are not permitted.

You must include the name of the table to be created. Any existing table of the same name will be dropped (deleted) and a new one created. If you omit the second argument, the value currently in $zTabSize will be used to set the number of columns.

By default, the initial value of $zTabSize is 10 (this can be changed in configure).

3.3.4 Added Builtin SQL Variables

3.3.4.1 $zsql

Returns the SQL server error message for the most recent command or 'ok.'
3.3.4.2 $zsqlOpen

Returns true if a connection to the SQL server is open, false otherwise.

3.3.4.3 $znative

Returns true if globals are being stored in the native global array

3.3.4.4 $zmysql

Returns true if globals are being stored in MySQL

3.3.4.5 $zpostgres

Returns true if globals are being stored in PostgreSQL

3.3.4.6 $ztable

Returns a comma separated string. The portion prior to the comma is the name of the most recently referenced table in which the Mumps globals are stored. The part after the comma is the maximum number of indices permitted in the table (same as $ztabsize).

3.3.4.7 $ztabsize

Returns the number of RDBMS columns available for global array indexes. May be set immediately prior to an sql/f command in which case the value in $ztabsize will be used to set the number of columns (range: 1 to 20) if a value is not explicitly given in the sql/f command.

3.3.4.8 $zsqlOutput

Contains the name of the file into which output from SQL commands will be written (see section 3.3.5). It may assigned a string value. This file name will be used by the SQL server to write output from sql commands. By default, the value in $zsqlOutput is the process id of the current program followed by the .tmp extension.

3.3.4.9 %globals()

Is an array containing the names of the Mumps tables in the relational database. Not currently available when using the native B-tree database. See also $globals section 2.2.1.1 on page 25. The following will list the available global array tables:

```mumps
for i=$order(%globals(i)) write i,!
```

3.3.5 SQL Command Output

If you execute a SQL command from Mumps that has output (for example, a SELECT command), the output will be written to a sequential file. The name of the file in which the results will be put is contained in the builtin variable $zsqlOutput. By default, the name in this variable is the Mumps program's Linux process ID followed by a .tmp extension.

For example:

```mumps
set $zsqlOutput="results.dat"
sql select * from tab;
open 1:"rssults.dat,old"
...
```

You may set $zsqlOutput to any valid file name and output will subsequently be directed to that file. Each new SQL command will overwrite any prior contents of the file whose name is in $zsqlOutput. Files created by SQL output may be read by your Mumps program.
4 Mixing Mumps and SQL Code

In some cases it can be much faster to extract data from the relational database with a single SQL command rather than with a series of iterative Mumps loops where each Mumps global array reference becomes a SQL SELECT statement.

For example, suppose we have a global array named \(^{\text{doc}}\) which contains a document-term matrix, commonly used in information storage and retrieval (IS&R) experiments. The global array represents documents and the words that occur in them.

In this global array, each row is a document designated by a number and the columns are word stems. The value in an array element indexed by a document number and a word stem is a score indicating how important the particular word stem is in the document. For example:

\[
\begin{align*}
^{\text{doc}}(411, "aortic") &= 32.32 \\
^{\text{doc}}(411, "appearance") &= 4.39 \\
^{\text{doc}}(411, "deteriorate") &= 5.26 \\
^{\text{doc}}(411, "hemodynamic") &= 4.47
\end{align*}
\]

Figure 4, taken from an information retrieval system operating on medical documents, gives an example of how we might print such a matrix.

```mumps
#!/usr/bin/mumps
for d=$order(^doc(d)) do
 . write "document ",d
 . for w=$order(^doc(d,w)) do
 .. write " ",w,"(","^doc(d,w),")"
 . write !
...
```

```mumps
document 411 aortic(32.32) appearance(4.39) deteriorate(5.26) hemodynamic(4.47)
document 446 antibiotic(4.92) bleed(4.39) bowel(4.16) cinedefecography(6.87)
document 458 added(4.39) antagonist(8.62) antagonize(11.54) atropine(32.69)
document 463 bile(8.46) bile-duct(17) biochemical(8.32) canalicular(6.18)
...
```

Figure 4 Mixing Mumps and SQL

In IS&R research we need a table of all the words used by all documents and, for each word, the number of documents in which it occurs.

We could do this in Mumps with the code shown in Figure 5 where we calculate the vector \(^{\text{df}}\) which gives, for each word, the number of documents in which the word occurs.

```mumps
for d=$order(^doc(d)) do
 . write "document ",d
 . for w=$order(^doc(d,w)) do
 .. if "$data(^\text{df}(w)) set ^\text{df}(w)=0
 .. else set ^\text{df}(w)=^\text{df}(w)+1
```

Figure 5 Expensive Iterative Mumps Code

However, in a relational database, this calculation generates a massive number of SQL queries. Each global array reference is a query! We could, however, create the same table with the SQL command shown in Figure 6:\footnote{The \ character causes line continuation beginning at the next non-blank or non-tab character.}

```sql
sql drop table if exists df;
if $\text{zsql}'=\text{ok}' write "$\text{Error dropping df },$\text{zsql},!$\text{halt}
```

\footnote{The \ character causes line continuation beginning at the next non-blank or non-tab character.}
sql create table df as \
    select a2 as a1, \n    trim(both from to_char(count(*),'9999999999')) as a2 from \n    doc group by a2;

if $zsql'="ok" write "Error creating df ",$zsql,! halt

Figure 6 SQL Equivalent

In Figure 6, the long create table command has been wrapped onto multiple lines for ease of reading.

In the SQL create table command we build the global array ^df consisting of a column of words (a1) and counts (a2) by selecting from ^doc words a2 from ^doc and the count of the number of time they occur.

The code:

    trim(both from to_char(count(*),'9999999999')) as a2

is how PostgreSQL\(^{16}\) converts the numeric result from count(*) to character string, the result of which is labeled a2 (Mumps prefers character data). The word, which is a2 in ^doc, becomes a1 in ^df.

By grouping the rows of ^doc by word (a2), we get a set of groups by word. The count function counts the number of elements in these groups and the result is a word and its count.

While this appears complicated, it isn't. It's fairly standard SQL code and efficient. Consequently, the construction of ^df is enormously quicker. The code from Figure 6 may be inserted into a larger Mumps program as shown in Figure 7. A sample of the output is shown\(^{17}\).

#!/usr/bin/mumps
for d=$order(^doc(d)) do
  . write "document ",d
  . for w=$order(^doc(d,w)) do
    .. write " ",w,"("","^doc(d,w),")"
  . write !

sql drop table if exists df;
if $zsql'="ok" write "Error dropping df ",$zsql,! halt

sql create table df as \
    select a2 as a1, \n    trim(both from to_char(count(*),'9999999999')) as a2 from \n    doc group by a2;

if $zsql'="ok" write "Error creating df ",$zsql,! halt

for w=$order(^df(w)) do
  . write w,?20,^df(w),!

.. document 411 aortic(32.32) appearance(4.39) deteriorate(5.26) hemodynamic(4.47)

\(^{16}\) MySQL has a simpler function to achieve the same result.
\(^{17}\) The for d=$order(...) format is peculiar to this version of Mumps. It was not part of any of the legacy standards.
Note also, the program in Figure 7 could have also been written as shown in Figure 8. In this version, a SQL select command is used to generate a file of output which is read and formatted by mumps. It is faster in that it involves fewer transactions with the server. Note that, by default, columns in the SQL output are separated by <tab> characters which appear as the $char(9) delimiter codes in the $piece() function.

```
#!/usr/bin/mumps
for d=$order(^doc(d)) do
  . write "document ",d
  for w=$order(^doc(d,w)) do
    .. write " ",w,"(",^doc(d,w),")"
  . write !
sql drop table if exists df;
if $zsql'="ok" write "Error dropping df ",$zsql,! halt
sql create table df as \
  select a2 as a1, \n    trim(both from to_char(count(*),'9999999999')) as a2 from \
    doc group by a2;
if $zsql'="ok" write "Error creating df ",$zsql,! halt
set $zsqlOutput="DocFreq.txt"
sql select * from df order by a1;
open 1:"DocFreq.txt,old"
for do
  . use 1 read line if '$test break
  . use 5
  . write $piece(line,$char(9),1),?20,$piece(line,$char(9),2),!
```
5 Implementation Notes

5.1 Notes on Arithmetic Precision

See section 1.12 on page 21 for additional details.

5.1.1 $fnumber()$

The built-in function $fnumber()$ only works on numbers that can be represented in a 64 bit floating point variable.

5.1.2 Exponential format numbers

All numbers represented in exponential format are treated as floating point numbers. If exponential format constants are used in expressions, they must be enclosed in quotes:

```plaintext
set i="1.23e3"*5
```

5.1.3 Arithmetic Precision

If found, Mumps will use the GNU bignum integer and MPFR floating point packages (this can be disabled by a configure option).

5.1.3.1 Floating Point Precision

When using extended precision MPFR numbers, floating point values have a default fractional precision of 72 bits. This can be changed with the --with-float-bits=val configure option. The maximum number of printed decimal digits is, by default, 20. This can be changed with the --with-float-digits=val configure option. The number of meaningful decimal digits that can be printed depends upon the number of bits in the fractional part of the floating point number. More bits mean more decimal digits can be printed.

If MPFR is not present, standard hardware double precision is used.

5.1.3.2 Integer Precision

There is no effective limit to integer precision except string length and memory when the extended precision bignum package is in use. Otherwise, precision is the same as the hardware long.

5.1.3.3 Performance

Extended precision arithmetic results in slower performance. The amount is dependent on how much arithmetic a program does, whether it is mainly integer or floating point (floating point is slower), and, in the case of fixed length numbers, how large the numbers are. Larger numbers result in slower computations.

5.1.4 Rounding

The $justify()$ function is useful to round lengthy repeating decimal floating point numbers to a more reasonable value.

5.2 Lock command with PostgreSQL or MySQL

Locks are not needed if using the PostgreSQL or MySQL for global array storage as SQL transaction commands can achieve the same effect. When using PostgreSQL or MySQL for the backend global array stores, the Lock should not be used. Instead, use the more modern native SQL transaction processing commands (BEGIN, COMMIT, ROLLBACK, etc.) to achieve the same effect with far greater integrity.

5.3 Lock command in client/server mode

In native B-tree mode, the Lock command creates a file named Mumps.Locks in /tmp where the lock information for the system is stored. If this file becomes corrupted due to abnormal terminations, it should be deleted. It will be rebuilt as needed.
5.4 Line Continuation

A line may be continued by placing a backslash at its end. The next line is appended beginning with the first non-blank or non-tab character. Note: this means that a blank must be on the prior line. Example:

```sql
create table df as select a2 as a1, trim(both from \
    to_char(count(*),'9999999999')) \  
as a2 from doc group by a2;
```

5.5 Naked indicator

This version of Mumps does not support the naked indicator. The naked indicator has no place in a modern or even semi-modern programming language. It was originally included in early versions of Mumps because of the inefficient binary mapping of an n-way tree which was used at the time to store the global arrays. The naked indicator was a short-hand to the interpreter to allow it to search for a global without stating at the top of the tree each time thus resulting in faster access. That is no longer the case with modern B-tree based access methods. Another issue is the perceived ambiguity of determining what exactly the naked indicator is after certain Mumps operations. Unfortunately, some legacy applications use it. These should be re-written.

5.6 Job command

The JOB command results in a C/C++ fork() function to be executed thus creating a child process. The child process will attempt to execute the argument to the JOB command. The JOB command may be used in native B-tree user mode but only one process may access the globals. In native client server mode, this restriction is not in effect. For PostgreSQL and MySQL, the child process should create a new connection.

The child process must end with a HALT command or the child process will hang.

5.7 File Names Containing Directory Information

When invoking a file name containing directory information (forward slash in Linux and backslash in DOS) with the DO or GOTO commands, the file name must be enclosed in quotes. For example:

```mumps
set x=""""^/home/user/xxx.mps"" goto @y
goto @""""^/home/user/xxx.mps""
```

Note the extra quotes. These are required.

5.8 File Names

File names should conform to variable naming conventions except that the first character of a file name may not be the per cent sign (%). The first character must be alphabetic. File names may only contain letters, digits and the per cent sign.

5.9 Array Index Collating Sequence

Array index collating sequences for both global and local array is ASCII. That is, for the $query() and $order() functions, all array indices will be presented in the same order as ASCII strings. Thus, in an array with 15 elements whose indices range from 1 to 15, the indices will be presented as:

```
1 10 11 12 13 14 15 2 3 4 5 6 7 8 9
```

Other versions of Mumps may present numeric indices in numeric order. This, however, leads to considerable inefficiencies in the data base.

You may achieve numeric ordering by storing the indices padded to left with blanks such as:

```mumps
for i=1:1:15 set ^a($justify(i,8))=i  
set i="" for set i=$order("a(i)) quit:i='' write +i," "
```

the indices will now be presented as:

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```
Note the the +i in the write command has the effect of converting the string to a number with no leading blanks.

5.10 Subroutine & Function Calls

Subroutines and functions may be performed in several ways as shown in Figure 9. Values returned from functions invoked by a do command are ignored. In standard Mumps, the $$ form is used only with function invocations.

Caution: be certain to include a halt or other exit in your program prior to any functions. If the halt is not present, function code will be entered and any passed variables will be undefined.

```mumps
#!/usr/bin/mumps
# calls.mps
set i=10
do fcn(i)
do fcn(5)
do $$fcn(i)
do $$fcn(5)
set k=$$fcn(5)
write "returned k=",k,!

set i=10
do fcn^ext.mps(i)
do fcn^ext.mps(5)
do $$fcn^ext.mps(i)
do $$fcn^ext.mps(5)
set k=$$fcn^ext.mps(5)
write "returned k=",k,!

do fcn^ext1.mps
do fcn^ext1.mps
do $$fcn^ext1.mps
do $$fcn^ext1.mps
set k=$$fcn^ext1.mps
write "returned k=",k,!

halt

fcn(x) write "in fcn(x) value passed is ",x,!
quit x
```

```
#!/usr/bin/mumps
# ext.mps
fcn(x) write "in fcn(x) value passed is ",x,!
quit x
```

```
#!/usr/bin/mumps
# ext1.mps
fcn write "in fcn ext1.mps",!
set x=22
quit x
```
output results:

in fcn(x) value passed is 10
in fcn(x) value passed is 5
in fcn(x) value passed is 10
in fcn(x) value passed is 5
in fcn(x) value passed is 5
returned k=5
in fcn(x) value passed is 10
in fcn(x) value passed is 5
in fcn(x) value passed is 10
in fcn(x) value passed is 5
in fcn(x) value passed is 5
returned k=5
in fcn ext1.mps
in fcn ext1.mps
in fcn ext1.mps
in fcn ext1.mps
in fcn ext1.mps
returned k=22

Figure 9 Subroutine/Function Calls

5.11 $fnumber() Function

The $fnumber() is implemented via the C function strfmon() which provides much greater flexibility when dealing with differing locales and, especially, currencies. The default locale is en_US.UTF-8 but this can be set with the configure option:

--with-locale=location-information

You may use $fnumber() with the legacy Mumps parameters or use it with a pattern parameter designed for strfmon().

If you use the strfmon() parameter option, the function takes two arguments. The first must be a number consisting of only numeric characters. The second is a character string conforming to a strfmon() pattern but preceded by an asterisk to distinguish the pattern from those used by the legacy Mumps function of the same name. The strfmon() function is well documented but here are some examples:

set x=12345.6789
write $fn(x,"*%!n")  ==> 12,345.68
write $fn(x,"*%n")  ==> $12,345.68
write $fn(x,"*%i")  ==> USD 12,345.68
write $fn(x,"*%n3")  ==> $12,345.683
write $fn(x,"*%20n")  ==> $12,345.68

5.12 $select() Function

All arguments of the $select() function are evaluated.

5.13 Compiling Large Programs

When compiling large programs, especially if MySQL is enabled, there may be a warning about variable tracking from the gcc/g++ compiler. You may ignore this.

5.14 Embedded Expressions

In several extended Mumps commands, the figure &~exp~ may appear. The expression exp is evaluated and the result replaces the figure. For example:

18 Using the compiler is not presently recommended.
set x="ls -lh"
shell &~x~

set x= "select * from abc;"
sql &~x~
6 Shell Command

6.1 shell

6.2 shell/g

6.3 shell/p

The shell command passes the remainder of the line to a shell for execution (sh in Linux). Shell output will appear on stdout. The command sets $test to false if the fork() fails, true otherwise.

This command is not presently available in the DOS version.

The shell/p form passes the remainder of the line to a shell for execution but opens a pipe from the shell to Mumps unit number 6. All stdout output from the shell is directed to unit number 6 and can be read with any of the input commands or functions in association with the use command.

The shell/g form passes the remainder of the line to a shell for execution (sh in Linux) and opens a pipe from the Mumps program to the shell as Mumps unit number 6. Data written to this unit becomes stdin to the shell. Output from the shell is written to stdout. Remember to close unit number 6 to signal end-of-file to the shell.

With no qualifier, the shell command passes the remainder of the command line to a shell. Input or output from the shell come from or go to stdin or stdout, respectively.

In all cases, the remainder of the command line is scanned for &~...~ expressions. The expression between &~ and ~ is evaluated and the result replaces the &~...~ expression.

For example:

| shell sort dictionary.tmp | uniq -c | sort -nr > dictionary.s |

The Linux shell created will do the following:

1. The file dictionary.tmp, a collection of words, will be sorted by sort and the output piped to uniq
2. uniq counts duplicate entries and pipes its output consisting of a count and a word to sort
3. sort sorts the result numerically by number of duplicates in reverse order and writes its output to dictionary.s.

| 1 shell/p sort dictionary.tmp | uniq -c | sort -nr |
| 2 open 1;"dictionary.s,new" |
| 3 for do |
| 4 . use 6 |
| 5 . read line |
| 6 . if '$test break |
| 7 . use 1 |
| 8 . write line,! |
| 9 close 1 |

Figure 10 Shell Command Example

The above does the same but the output will be presented to Mumps unit 6 which reads and writes the result to the file named dictionary.s
7 Added Commands

7.1 Database expr

The database command may be used to set the name of the files to be used to store the native global arrays. The expression will be evaluated and the resulting name will become the name, suffixed .key and .dat, of the files in which the native global arrays are stored. The expression may contain directory information. For example:

```mumps
database "/home/user/data/mumps"
```

will cause the system to access files:

```
/home/user/data/mumps.key
/home/user/data/mumps.dat
```

for the global array tree and data files. If directory information is omitted, the files will be in the current directory.

This command must be issued prior to any attempt to access the global arrays. It only works with the native B-tree database option.

7.2 Zhalt return_code

The zhalt command will terminate the current program with a return error code given by its argument. Example:

```mumps
if a=0 zhalt 99
```

The value of $? in the BASH environment will be 99.
8 Z Functions and System Variables

$zfunctions$ are extensions added by the implementor and not covered by the standard. Thus, many if not all of the following M2 extensions may not be supported or supported differently in other implementations. Likewise, there are implementer defined system variables which may be queried and, in some cases, set.

M2 implementation note: you may add new $z$ functions by modifying the function $zfcn()$ located in the source file $bifs.cpp.in$

8.1 System Variables

8.1.1 $zttable$

The value in $zttable$ is the name of the current database table if you are using PostgreSQL or MySQL to store the global arrays. It has no meaning if you are using the native Btree. The default value will be $mumps$ unless this was changed during the $configure$ step or changed during execution.

You may set the name of the database table in use by setting $zttable$. All database references will take place in the table you set $zttable$ to until you change it again or the program terminates.

When you start a Mumps program, $zttable$ reverts to the value set by $configure$.

The string returned by $zttable$ consists of two parts separated by a comma. The first part is the name of the table and the second part is the number of columns available for global array indices. The number is indices is two less than the actual number of columns as one column is reserved for the global array name named $gbl$, and one column for the value stored at the global reference named $ax$ where the value of $x$ is one greater that the number of indices.

If you change the default table, you need to insure that it exists and is properly defined for use my Mumps. The $sql/f$ command will create and initialize a new table or re-initialize to empty an existing table. Warning messages may appear the first time you create a table.

8.1.2 $ztTabSize$

The maximum number of indices permitted in a global array reference. When queried, this variable returns the current setting. It may be set (maximum of 20). If you set it, you must initialize the global array before using it or errors will result.

8.1.3 $zProgram$

Returns a string with the name of the currently executing program.

8.2 Math Functions

The following C/C++ math functions are available in M2. Their arguments and return values are the same as the correspondingly named C++ functions.

8.2.1 $zabs(arg)$ absolute value

Function returns the absolute value of its numeric argument.

8.2.2 $zacos(arg)$ arc cosine

Computes the inverse cosine (arc cosine) of the input value. Arguments must be in the range -1 to 1.

8.2.3 $zasin(arg)$ Arc sine

Computes the inverse sine (arc sine) of the argument $arg$. Arguments must be in the range -1 to 1.
8.2.4 $atan(arg)$ Arc tangent
Computes the inverse tangent (arc tangent) of the input value.

8.2.5 $zcos(arg)$ Cosine
Computes the cosine of the argument arg. Angles are specified in radians.

8.2.6 $zexp(arg)$ Exponential
Calculates the exponential of arg, that is, $e$ raised to the power arg (where $e$ is the base of the natural system of logarithms, approximately 2.71828).

8.2.7 $zexp2(arg)$ Exponential base 2
Calculates 2 raised to the power arg.

8.2.8 $zexp10(arg)$ Exponential base 10
Calculates 10 raised to the power arg.

8.2.9 $zlog(arg)$ Natural log
Returns the natural logarithm of arg, that is, its logarithm base $e$ (where $e$ is the base of the natural system of logarithms, 2.71828...).

8.2.10 $zlog2(arg)$ Base 2 log
Returns the base 2 logarithm of arg.

8.2.11 $zlog10(arg)$ Base 10 log
Returns the base 10 logarithm of arg.

8.2.12 $zpow(arg1,arg2)$ Power function
Calculates arg1 raised to the exponent arg2.

8.2.13 $zsqrt(arg)$ Square root
Function returns the square root of its numeric argument.

8.2.14 $zsinsin(arg)$ Sine function
Computes the sine of the argument arg. Angles are specified in radians.

8.2.15 $zthtan(arg)$ Tangent function
Computes the tangent of arg.

8.3 Date functions

8.3.1 $zdate(or $zd ) formatted date string
Function returns the system date and time in standard system printable format. This includes: day of week, month, day of month, time (hour:minute:second), and year (4 digits).

8.3.2 $zd1$ numeric internal date
Returns the number of seconds since January 1, 1970 - a standard used in Linux. This number may be used to accurately correlate events.

8.3.3 $zd2(InternalDate)$ date conversion
Translates the Linux time from $ZD1 into standard system printable format. The argument is a Linux format time value.

8.3.4 $zd3(Year,Month,Day)$ Julian date
Returns the day of the year (Julian date) for the Gregorian date argument.
8.3.5 \$zd4(Year,DayOfYear) Julian to Gregorian

Returns the Gregorian date for the Julian date argument.

8.3.6 \$zd5(Year, Month, Day) comma listed date

Returns a string consisting of the year, a comma, the day of year, and the number of days since Sunday (Monday is 1).

8.3.7 \$zd6 hour:minute

Returns a string consisting of the hour, a colon, and the minute.

8.3.8 \$zd7 hyphenated date

Returns a string consisting of the year, hyphen, month, hyphen, and day of month. If an argument is given in the form of the number of seconds since Jan 1, 1970, the result returned will reflect the argument date.

8.3.9 \$zd8 hyphenated date with time

Returns a string consisting of the year, hyphen, month, hyphen, and day of month, comma, and time in HH:MM format. If an argument is given in the form of the number of seconds since Jan 1, 1970, the result returned will reflect the argument date.

8.4 Special Purpose Functions

The following special purpose functions are available:

8.4.1 \$zb(arg) remove blanks

Function returns a string in which all leading blanks have been removed and all multiple blanks have been replaced by single blanks. See also \$zNoBlanks(). Figure 11 gives examples.

```
1 #!/usr/bin/mumps
2 set a="   abc   xyz     123    
3 write $zb(a),"***",!
output:
abc  xyz 123  ***
```

Figure 11 $zb() Examples

8.4.2 \$zchdir(directory_path) change directory

Function changes the current directory to the path specified. If the operation succeeds, a zero is returned. If it fails, -1 is returned.

8.4.3 \$zCurrentFile Current Mumps File

Returns the name of the currently executing Mumps program file (if any) or blank.

8.4.4 \$zdump[(filename)] dump global arrays

Function dumps the globals to a sequential ASCII file in the current directory. If an argument is given, it is taken as the name of the file to which the globals will be written. If the argument is omitted, a file name is constructed from the system date of the form number.dmp where number is the value of the C++ time() function at the time of the dump.

The dump file is a pure ASCII text file. Each entry in the global array is represented by two lines. The first line is the global array reference and the second line is the store value. In the global array reference, parentheses and commas are replaced by the "~" character. Thus, if you wish to use this facility, you may not include the "~" character in a global array index.

The function \$zrestore() reloads the global arrays from a dump file (see below).
$zdump and $zrestore do not work when PostgreSQL is used for the global array store.

8.4.5 $zrestore[(arg)] restore globals

Function restores the globals from a dump file produced by $zdump. If an argument is given, it is taken as the name of the dump file otherwise, the default name dump is used.

$zdump and $zrestore do not work when PostgreSQL is used for the global array store.

8.4.6 $zfile(arg) file exists test

Function returns a zero or one indicating if the file given as the argument exists.

8.4.7 $zflush flush Btree buffers

Function flushes all modified native global array handler buffers to disk. The function should only be used with the native globals. After flushing, all updates to the btree file system have been committed. In cases where the internal buffers are very large, this function may take several seconds to execute. The function returns the empty string. Flushing the buffers is a precaution against system failure which would otherwise result in corruption of the global arrays.

8.4.8 $zgetenv(arg) get environment variable

Returns the contents of the environment variable specified as arg or the empty string if the variable is not found.

8.4.9 $zhtml(arg) encode HTML string

Function encodes its argument in the form necessary to be a cgi-bin parameter. That is, alphabetics remain unchanged, blanks become plus signs and all other characters become hexadecimal values, preceded by a percent sign.

8.4.10 $zhit global array cache hit ratio

Function calculates and returns the native global array cache hit ratio. This number ranges between zero and one. A value of one indicates all requests were satisfied from the cache while a value of zero indicates no requests were satisfied from the cache. Calling this function resets the hit ratio to zero. A higher value for the hit ratio indicates better database performance.

8.4.11 $zlower(string) convert to lower case

Function returns the input string with alphabetics converted to lower case.

8.4.12 $znormal(arg1[,arg2]) word normalization

Function converts the word passed as argument 1 to lower case and removes any embedded punctuation. If a second argument is given, the word is truncated to the length specified by this argument. If no second argument is given, words are truncated to 25 characters if their length exceeds 25 characters.

8.4.13 $zNoBlanks(arg) remove all blanks

Returns arg with all blanks removed. See also: $zb.

8.4.14 $zpad(arg1,arg2) left justify with padding

Function left justifies the first argument in a string whose length is given by the second argument, padding to the right with blanks.

8.4.15 $zseek(arg)

Function takes one argument (a positive integer) which is a byte offset in the currently active (use) file. The command moves the file pointer to that location in the file. $zseek() may only be used on files opened with old attribute. Figure 12 gives examples.

1. #!/usr/bin/mumps
2. open 1;"tdb,new"
3. for j=1:1:1000 do
4. use 1
5. set i=$ztell
6. set ^a(j)=i
7. write "**** ",j,!
8
9 close 1
10 open 1:"tdb,old"
11 for j="":$order(^a(j)):"" do
12 . use 1
13 . set i=$zseek(^a(j))
14 . read a
15 . use 5
16 . write a,!

output:

**** 1
**** 10
**** 100
**** 1000
**** 101
**** 102
**** 103
**** 104
**** 105
**** 106
**** 107
**** 108
**** 109
**** 11
**** 110
**** 111
...

Figure 12 $Zseek() Examples

8.4.16 $zsrand(arg)

Seed the random number generator. The value passed as the argument will seed the internal random number generator. If the random number generator is re-seeded with the same seed, the sequence of random numbers produced by $random will be the same. The value passed must be a positive integer.

8.4.17 $zstem(arg)

Returns an word English word stem of the argument. This function attempts to remove common endings from words and return a root stem.

8.4.18 $zsystem(arg)

Executes "arg" in a system shell. Returns -1 (fork failed) or the return code of the execution of the argument. See also the shell command.

8.4.19 $ztell

Function returns the byte offset in the currently open file. Similar to the C++ ftello function. Note: The offset returned is for the file most recently made the default i/o file by the use command. $ztell may be used on either a file opened as new, old or append. (See example under $zseek above)

8.4.20 $zu(expression)

Function returns 1 if the expression is numeric, 0 otherwise.
8.4.21 $zwi(arg)

Function loads an internal buffer with the string given as the argument. The alphabetic characters of the argument are converted to lower case. The contents of this buffer are returned by the $zwn and $zwp functions. Figure 13 gives examples.

8.4.22 $zwn extract words from buffer

Function returns successive words from the internal buffer delimited by blanks. When no more words remain, it returns an empty string (string of length zero). Returned words are converted to lower case. See $zwi.

8.4.23 $zwp extract words from buffer

Function returns successive words from an internal buffer delimited by blanks and punctuation characters. When no more words remain, it returns an empty string (string of length 0). Returned words are converted to lower case. See $zwi.

8.4.24 $zws(string) initialize internal buffer

Initializes the parse buffer but does not convert "string" to lower case as is the case with $zwi.

```mumps
!#!/usr/bin/mumps
2 set i="now, is the time, for all good"
3 set %=$zwi(i)
4 for w=$zwp write w,!
5 write "----",!
6 set %=$zwi(i)
7 for w=$zwn write w,!
```

**output:**

```
now,  
is  
the  
time  
,  
for  
all  
good  
-----
now,  
is  
the  
time,  
for  
all  
good
```

Figure 13 $Zwi() Examples

8.4.25 Scan Functions

8.4.25.1.1 $zzScan

8.4.25.1.2 $zzScanAlnum

8.4.25.1.3 $zzInput(var)
The functions return the next word in the current input stream delimited by white space. Words are restricted to a maximum length of 1023. Successive calls return successive words. When there are no more input words, an empty string is returned and $test$ is set to false.

If only part of a line is scanned as a result of these functions, a subsequent read command will begin at the white space following the last word returned.

If scanning input from stdin (i/o unit 5), you may signal end of file with a control-d on a separate line by itself. This will result terminate the scan and $test$ will be set to false.

$zzScan$ returns all words delimited by whitespace with no conversion. Words may contain any printable ASCII character.

$zzScanAlnum$ processes words before returning them according to the following rules:

- Special characters at the beginning of a word are ignored.
- Words beginning with digits are not returned. If a word begins with one or more special characters followed by a digit, it is not returned.
- Words shorter than 3 characters or longer than 25 characters are not returned.
- Words are converted to all lower case characters.
- If a word contains embedded special characters, it is treated as a delimiter.

Both functions will advance to additional lines as needed. If a word exceeds 1023 bytes, the results are undefined. See Figure 14 for an example.
$zzInput(var)$ reads an entire input line, converts all characters to lower case, separates the words, removes punctuation (as defined by the C $ispunct()$ function except hyphen), and stores the words into a numerically indexed array whose name is the value of the variable or constant passed as the argument. The function returns the number of elements in the array. A return of zero indicates no input was obtained (end of file). As the array created by the function could be quite large, you should probably $kill$ it when no it is longer needed. The maximum line length permitted is twice the system parameter $MAX_STR$ (9,000 bytes by default).

8.5 Vector and Matrix Functions

8.5.1 $zzAvg(vector)$

Computes and returns the average of the numeric values in the vector. For example, see Figure 15.

```
#!/usr/bin/mumps
for i=1:1:10 set ^a(99,i)=i
set i=$zzAvg(^a(99))
write "average=",i,!
```

The above writes 5.5

8.5.2 $zzCentroid(gblMatrix,gblRef)$

A centroid vector $gblRef$ is calculated for the invoking two dimensional global array $gblMatrix$. The centroid vector is the average value for each for each column of the matrix. Any previous contents of the global array named to receive the centroid vector are lost. The global array $gblMatrix$ must contain at least two dimensions. See Figure 16 for an example. The matrix must be a top level global array.

```
#!/usr/bin/mumps
for i=0:1:10 do
  . for j=1:1:10 do
    .. set ^A(i,j)=5
  set ^=zzCentroid(^A,^B)
for i=1:1:10 write ^B(i),!
```

output:

<table>
<thead>
<tr>
<th>5</th>
<th>5</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

8.5.3 $zzCount(gblVector)$

Computes and returns the number of numeric values in the vector and any descendants. For example:

```
#!/usr/bin/mumps
```
2   kill ^a
3   for i=1:1:10 set ^a(99,i)=i
4     set i=$zzCount(^a(99))
5     write "count=",i,!

Figure 17 $zzCount() Example

The above writes 10

8.5.4 $zzMax(gbl)

Computes and returns the maximum numeric value in the vector and any descendants. See Figure 18 for an example.

1    #!/usr/bin/mumps
1      for i=1:1:10 set ^a(99,i)=i
2      set i=$zzMax(^a(99))
3      write "max=",i,!

output:

10

Figure 18 $zzMax() Example

The above writes the largest value stored in the vector.

8.5.5 $zzMin(gbl)

Returns the minimum numeric value stored in the vector and any descendants. See Figure 19 for an example.

1    #!/usr/bin/mumps
1      for i=1:1:10 set ^a(99,i)=i*2
2      set i=$zzMin(^a(99))
3      write "min=",i,!

output:

2

Figure 19 $zzMin() Example

8.5.6 $zzMultiply(gbl1,gbl2,gbl3)

Multiplies the first and second matrix leaving the result in the third. The ordinary rules of algebra apply. Figure 23 gives an example. The arguments gbl1 and gbl2 must be top level, two dimensional arrays.

8.5.7 $zzSum(gblVector)

Computes and returns the sum of the numeric values stored in the vector. For example, see Figure 24.

8.5.8 $zzTranspose(gblMatrix1,gblMatrix2)

Transposes the first global array matrix leaving the result in the second. For example, see Figure 25. the argument gblMatrix1 must be a top level, two dimensional array.

8.6 Text Processing Functions

The following functions are used in connection with experiments in information storage and retrieval.
8.6.1 Similarity Functions

8.6.1.1 $zzCosine(gbl1,gbl2)$

8.6.1.2 $zzSim1(gbl1,gbl2)$

8.6.1.3 $zzDice(gbl1,gbl2)$

8.6.1.4 $zzJaccard(gbl1,gbl2)$

These compute the Cosine, Sim1, Dice and Jaccard similarity coefficients between document vectors given as the first and second arguments. Both arguments are numeric global array vectors. The formulae are given in Figure 20 and an example in code is given in Figure 21. The formulae calculate the similarities between two global array vector $gbl1$ and global array vector $gbl2$. The vectors need not be of equal length. Missing elements are interpreted as zero. The vectors should be top level vectors.

$$
Simularity_{Dice}(i, j) = \frac{2 \sum_{k=1}^{t} Term_{ik} \cdot Term_{jk}}{\sum_{k=1}^{t} Term_{ik} + \sum_{k=1}^{t} Term_{jk}}
$$

$$
Simularity_{Jaccard}(i, j) = \frac{\sum_{k=1}^{t} Term_{ik} \cdot Term_{jk}}{\sum_{k=1}^{t} Term_{ik} + \sum_{k=1}^{t} Term_{jk} - \sum_{k=1}^{t} (Term_{ik} \cdot Term_{jk})}
$$

$$
Simularity_{Cosine}(i, j) = \frac{\sum_{k=1}^{t} Term_{ik} \cdot Term_{jk}}{\sqrt{\sum_{k=1}^{t} Term_{ik}^2 \cdot \sum_{k=1}^{t} Term_{jk}^2}}
$$

$$
Simularity_{Sim1}(i, j) = \sum_{k=1}^{t} Term_{ik} \cdot Term_{jk}
$$

Figure 20 Similarity Formulae

```
#!/usr/bin/mumps
kill ^A
kill ^B
set ^A("1")=3
set ^A("2")=2
set ^A("3")=1
set ^A("4")=0
set ^A("5")=0
set ^A("6")=0
set ^A("7")=1
set ^A("8")=1
set ^B("1")=1
set ^B("2")=1
set ^B("3")=1
set ^B("4")=0
```
set ^B("5")=0
set ^B("6")=1
set ^B("7")=0
set ^B("8")=0
write "Cosine=",$zzCosine(^A,^B),!
write "Sim1=",$zzSim1(^A,^B),!
write "Dice=",$zzDice(^A,^B),!
write "Jaccard=",$zzJaccard(^A,^B),!

output:
Cosine=0.75
Sim1=6
Dice=1
Jaccard=1

8.6.2 $zzBMGSearch(arg1,arg2)

Boyer-Moore-Gosper Function returns the number of non-overlapping occurrences of arg1 in arg2.

These functions, were obtained from
and were written by Jeffrey Mogul (Stanford University), based on code written by James A. Woods (NASA Ames, an agency of the U.S. Government) and are thus believed to be in the public domain.
Figure 22 gives an example.

1 #!/usr/bin/mumps
2 set key="now"
3 set str="now is the now of the now in the know"
4 write $zBMGSearch(key,str),!

output:
4

8.6.3 $zPerlMatch(string,pattern)

Applies the Perl pattern to string and returns 1 if the pattern fits and 0 otherwise. The $zPerlMatch function has the side effect of creating variables in the local symbol table to hold backreferences, the equivalent concept of $1, $2, $3, ... in Perl. Up to nine backreferences are currently supported, and can be accessed through the same naming scheme as Perl ($1 through $9). These variables remain defined up to a subsequent call to $zPerlMatch, at which point they are replaced by the backreferences captured from that invocation. Undefined backreferences are cleared between invocations; that is, if a match operation captured five backreferences, then $6 through $9 will contain the empty string. Figure 26 contains examples (long lines wrapped).
10 set ^e("1","2")=-2
11 set ^e("1","3")=4
12 set ^e("1","4")=7
13 set ^e("2","1")=-6
14 set ^e("2","2")=1
15 set ^e("2","3")=-3
16 set ^e("2","4")=0
17 set %=zzMultiply(^d,^e,^f)
18 for i="":$order(^f(i)):"" do
19   for j="":$order(^f(i,j)):"" do
20     .. write i," ",j," ",^f(i,j),!
21
output:

   1 1  -8
   1 2   -1
   1 3   -1
   1 4    14
   2 1    11
   2 2   -3
   2 3    7
   2 4    7
   3 1   -24
   3 2    4
   3 3  -12
   3 4     0

Figure 23 $zzMultiply() Example

1 #!/usr/bin/mumps
2 for i=1:1:10 set ^a(99,i)=i
3 set i=zzSum(^a(99))
4 write "sum=",i,!

output:

   55

Figure 24 $zzSum() Example

1 #!/usr/bin/mumps
2 kill "f"
3
4 set ^d("1","1")=2
5 set ^d("1","2")=3
6 set ^d("2","1")=4
7 set ^d("2","2")=0
8
9 set %=zzTranspose(^d,^f)
10 for i="":$order(^f(i)):"" do
11   for j="":$order(^f(i,j)):"" do
12     .. write i," ",j," ",^f(i,j),!
13
output:

   1 1 2
   1 2 4
Figure 25 $zzTranspose() Example

1 #!/usr/bin/mumps
2 write "Please enter a telephone number:",!
3 read phonenum
4 set p="^(1-)?(\(\d{3}\))?(­| )?\d{3}­?\d{4}$"
5 if $zperlmatch(phonenum,p) do
6   . write "+++ This looks like a phone number.",!
7 else do
8   . write "--- This didn't look like a phone number.",!
9 output:
10 Please enter a telephone number:
11 (123) 456-7890
12 +++ This looks like a phone number.
13 The area code is: (123)
14 Please enter a telephone number:
15 (123) 456-7890
16 +++ This looks like a phone number.

Figure 26 $zPerlMatch() Example

8.6.4 $zReplace(string,pattern,replacement)

The regular expression in pattern is evaluated on string and, if there is a match, the matching section is replaced by replacement. Figure 27 contains an example. In the first part, the word 'is' is replaced by 'IS'. In the second part, a match is sought for any content between two sets of matching brackets ([...]). The matched section is in back reference $2. This is then used as a pattern to be replaced.

8.6.5 $zShred(string,length)

8.6.6 $zShredQuery(string,length)

The $zShred() function segments the input argument string into fragments of length size upon successive calls. The function returns a string of length zero when there are no more fragments of size length remaining (thus, short fragments at the end of a string are not returned).

$zShred copies the input string to an internal buffer upon the first call. Subsequent calls retrieve from this buffer. When the buffer is consumed, the function will copy the contents of the next string submitted to the buffer. Figure 28 contains an example.

1 #!/usr/bin/mumps
2 set a="now is the time for all"
3 set a=$zReplace(a,"is","IS")
4 write a,!
5 set a="[[now is the time]]"
6 if $zPerlMatch(a,"\([\[\]()\.*\]\])\)" do
7   . set a=$zReplace(a,$2,"ABC")
8   . write a,!
9 output:
10 now IS the time for all
Figure 27 $zReplace()$ Example

```
#!/usr/bin/mumps
set a="now is the time for all good men to "
set a=a_"come to the aid of the party"
for do quit:j=""
  set j=$zShred(a,5)
  if j="" quit
  write j,!
```

```
output:
nowis
theti
mefor
algo
odmen
tocom
etoth
eaido
fthepe
```

Figure 28 $zShred()$ Example

The $szShredQuery$ function segments length shifted copies of the input string into fragments of size length upon successive calls. That is, the function first returns all the fragments of size length of the string in the same manner as $szShred$. However, it then shifts the starting point of the input string to the right by one and returns all the fragments of size length relative to the shifted starting point. If repeatedly called, it repeats this process a total of length times. When there are no more combinations, the empty string is returned as shown in Figure 29.

```
#!/usr/bin/mumps
set a="now is the time for all good men to come to "
set a=a_"the aid of the party"
for do quit:j=""
  set j=$szShredQuery(a,5)
  if j="" quit
  write j,!
```

```
output:
nowis
tothe
goodm
theti
aidof
entoc
mefor
thepa
ometo
algo
wisth
theai
odmen
etime
dofth
tocom
foral
epart
etoth
lgood
isthe
eaido
mento
timef
fthepe
comet
orall
owist
othea
goodm
hetime
idoft
entoc
efors
hepar
ometo
llgoo
isthe
theai
dment
timef
dofth
ocene
orall
epart
```

Figure 29 $ShredQuery()$ Example
8.6.7 $zzSoundex(s1)

Returns the Soundex code for the argument string as follows:

1. All letters are converted to lower case;
2. Non-alphabetic characters are removed;
3. Adjacent duplicate letters are replaced by a single occurrence;
4. The first letter is retained;
5. The letters b, f, p, and v are replaced by the number 1;
6. The letters c, g, j, k, q, s, x, and z are replaced by the number 2;
7. The letters d and t are replaced by the number 3;
8. The letter l is replaced by the number 4;
9. The letters m and n are replaced by the letter 5;
10. the letter r is replaced by the number 6;
11. The is truncated to four characters.

8.6.8 $zSmithWaterman(s1,s2,algn,mat,gap,noMatch,match)

Computes the Smith Waterman score between two strings. Result returned is the highest alignment score achieved. String lengths are limited by STR_MAX in the interpreter. If you compare very long strings (>100,000 characters), you may exceed stack space. This can be increased under Linux with the command:

```bash
ulimit -s unlimited
```

Figure 30 gives an example.

```mumps
#!/usr/bin/mumps
set s1="now is the time"
set s2="now i th time"
set i=$zSmithWaterman(s1,s2,1,0,-1,-1,2)
write "score=",i,!
```

**output:**

```
1 now- is the time 16
::: ::: ::: ::::
1 now  i- th time 16
```

**score=23**

Figure 30 $zSmithWaterman() Example

Parameters:

If algn is zero, no printout of alignments is produced. If algn is not zero, a summary of the alternative alignments will be printed.

If mat is zero, intermediate matrices will not be printed.

The parameters gap, noMatch and match are the gap and mismatch penalties (negative integers) and the match reward (a positive integer).

If insufficient memory is available, a segmentation violation will be raised. Try increasing your stack size.

8.6.9 $zzIDF(global,doccount)

Calculates the Inverse Document Frequency score of words contained in the argument global. The parameter doccount is the total number of documents. The index of each element of the global vector is a word and the value stored is the number of times the word occurs in the collection. Figure 31 gives an example. The vector argument global must be a top level array.
Calculates the Term-Term co-occurrence matrix for the Document-Term matrix in \textit{global1}. The result is placed in \textit{global2}.

A Term-Term matrix has terms (words) as the indices of its rows and columns. A Term-Term matrix gives, for each position, the degree to which the term corresponding to the row is similar to the term corresponding to the column. The diagonal, which is the degree a term is related to itself, is ignored. Both operands must be top level arrays.

In both the doc-doc and term-term matrices, the upper and lower diagonal matrices are mirror images of one another. Figure 32 gives an example. The order of words in the output will depend upon which data base facility is in use and what it's collating settings are. The Native global array handler collates according to ASCII-7.
27 for i="":$order(^B(i)):"" do
28   . write i,!
29   . for j="":$order(^B(i,j)):"" do
30     .. write ?10,j,"",^B(i,j),!

| output: |
|------------------|------------------|------------------|
| USB              | computer 1       | monitor 1        |
|                  | disk 1           | printer 1        |
| laptop 1        | computer 1       | language         |
| memory 1        | laptop 1         | data 1           |
| printer 1       | memory 1         | program 1        |
| computer        | USB 1            | printer 1        |
| data 1          | laptop           | USB 1            |
| disk 2          | language 1       | computer 2       |
| laptop 3        | disk 2           | memory 2         |
| memory 2        | monitor 1        | program 1        |
| printer 2       | language 1       | program 1        |
| program 2       | memory 2         | program 1        |
|                  | laptop 2         | printer 2        |
|                  | computer 2       | data 1           |
|                  | laptop           | language 1       |
|                  | disk 1           | memory 1         |
|                  | monitor 1        | printer 1        |
|                  | memory 1         | memory 1         |
|                  | printer 1        | program 1        |

Figure 32 $zTermCorrelate() Example

8.6.10.2 $zzDocCorrelate(gblref1,gblref2,mthd,thrshld)

A square Document-Document matrix $gblref2$ is calculated from the Document-Term matrix $gblref1$ according to method $mthd$ (Cosine, Sim1, Dice, Jaccard). The value of elements in the Document-Document matrix will not exceed threshold ($thrshld$) and the cells associated with corresponding document numbers will not exist.

A Document-Document matrix has document id’s as its row and column indices. A cell in the matrix indicates the degree to which the row document is related to the column document. The diagonal is ignored. Figure 33 gives an example.

8.6.11 Stop and Synonym Functions

8.6.11.1 $zStopInit(arg)$

8.6.11.2 $zStopLookup(word)$

8.6.11.3 $zSynInit(fileName)$

8.6.11.4 $zSynLookup(word)$

A call to $zStopInit(file_name)$ will open and load a file of stop words into a C++ container. The file should consist of one word per line. If the file cannot be opened or there is insufficient memory to hold the list of words, the program will halt with an error message. $zStopInit()$ converts all words to lower case.
set ^A("1","computer")=5
set ^A("1","data")=2
set ^A("1","program")=6
set ^A("1","disk")=3
set ^A("1","laptop")=7
set ^A("1","monitor")=1

set ^A("2","computer")=5
set ^A("2","printer")=2
set ^A("2","program")=6
set ^A("2","memory")=3
set ^A("2","laptop")=7
set ^A("2","language")=1

set ^A("3","computer")=5
set ^A("3","printer")=2
set ^A("3","disk")=6
set ^A("3","memory")=3
set ^A("3","laptop")=7
set ^A("3","USB")=1

set %=$zzDocCorrelate(^A,^B,"Cosine",.5)

for i="":$order(^B(i)):"" do
write i,!
for j="":$order(^B(i,j)):"" do
write ?10,j,"",^B(i,j),!

output:

1
2 0.887096774193548
3 0.741935483870968

2
1 0.887096774193548
3 0.701612903225806

3
1 0.741935483870968
2 0.701612903225806

Figure 33 $zDocCorrelate()Example

A call to $zStopLookup(word) will return 1 if word is in the stop list, 0 otherwise. Words presented to $zStopLookup(word) should be in lower case.

$SynInit() opens a synonym file. The file should consist of two or more words per line separated by from one another by one blank. The words are treated as synonyms with the first word on each line as the primary synonym. The primary synonym may be a code or category number. This word or code will be returned if any of the remaining words are passed as arguments to $SynLookup(). Figure 34 gives an example.

8.7 SQL functions

These functions are peculiar to this implementation.

8.7.1 $zsql

Returns the SQL server error message for the most recent command or 'ok.'
8.7.2 $zsqlCols

Returns a string consisting of the columns names for the most recent operation that returned
tuples. Each name is separated from the next by a TAB character ($char(9)).

Assume that the file “stop” contains the word “and”

```mumps
set %=zStopInit("stop")
if %zStopLookup("and") write "yes",!
```

Writes yes

Assume that the file “synonyms” contains a line with the text:

```
compression compressions compress compressed compresses
```

```mumps
set %=zSynInit("synonyms")
write %zSynLookup("compressions"),!
```

Output:

```
compression
```

**Figure 34 Stop List Functions**

8.7.3 $zsqlOpen

Returns true if a connection to the SQL server is open, false otherwise.

8.7.4 $zNative

$znative returns true if globals are being stored in the native global array.

8.7.5 $zMysql

$zmysql returns true if globals are being stored in MySQL

8.7.6 $zPostgres

$zpostgres returns true if globals are being stored in PostgreSQL

8.7.6.1 $zTable

$ztable returns a comma separated string. The portion prior to the comma is the current RDBMS
table in which the Mumps globals are stored. The part after the comma is the maximum number of
indices permitted in the table (same as $ztabsize).

$ztable may be set. If it is set immediately prior to an sql/f command, it is the name of the table to
be created and/or initialized which now becomes the default table.

If $ztable is set to the name of a table which exists, global array reference will be made to this table
until $ztable is changed. When $ztable is changed to the name of an existing table, $ztabsize is
updated to the value for the now current table.

8.7.7 $zTabsize

$ztabsize returns the number of RDBMS columns available for global array indexes. May be set
immediately prior to an sql/f command in which case the value in $ztabsize will be used to set the
number of columns (range: 1 to 20).

8.7.8 $zsqlOutput

$zsqlOutput contains the name of the file to which output from SQL commands will be written. It
may assigned a string value that will be used as the SQL command processor output file by the next
SQL command.
9 Pattern Matching

9.1 Mumps 95 Pattern Matching

Author: Matthew Lockner

Mumps 95 compliant pattern matching (the '?' operator) is implemented in this compiler/interpreter as given by the following grammar:

\[
\begin{align*}
\text{pattern} & \ ::= \ \{\text{pattern} \ \_\atom\} \\
\text{pattern} \ _\atom & \ ::= \ \text{count} \ \text{pattern} \ _\element \\
\text{count} & \ ::= \ \text{int} | \ . | \ . \ \text{int} | \ \text{int} \ . | \ \text{int} \ . \ \text{int} \\
\text{pattern} \ _\element & \ ::= \ \text{pattern} \ _\code \ \{\text{pattern} \ _\code\} \ | \ \text{string} \ | \ \text{alternation} \\
\text{pattern} \ _\code & \ ::= \ \text{A} | \ \text{C} | \ \text{E} | \ \text{L} | \ \text{N} | \ \text{P} | \ \text{U} \\
\text{alternation} & \ ::= \ (\ \text{pattern} \ _\atom \ \{,\ \text{pattern} \ _\atom\} \ )
\end{align*}
\]

The largest difference between the current and previous standard is the introduction of the alternation construct, an extension that works as in other popular regular expressions implementations. It allows for one of many possible pattern fragments to match a given portion of subject text.

A string literal must be quoted. Also note that alternations are only allowed to contain pattern atoms and not full patterns; while this is a possible shortcoming, it is in accordance with the standard. It is a trivial matter to extend alternations to the ability to contain full patterns, and this may be implemented upon sufficient demand.

Pattern matching is supported by the Perl-Compatible Regular Expressions library (PCRE). Mumps patterns are translated via a recursive-descent parser in the Mumps library into a form consistent with Perl regular expressions, where PCRE then does the actual work of matching. Internally, much of this translation is simple character-level transliteration (substituting '|' for the comma in alternation lists, for example). Pattern code sequences are supported using the POSIX character classes supported in PCRE and are mostly intuitive, with the possible exception of 'E', which is substituted with `[[:print][:cntrl:]]. Currently, this construct should cover the ASCII 7-bit character set (lower ASCII).

Due to the heavy string-handling requirements of the pattern translation process, this module uses a separate set of string-handling functions built on top of the C standard string functions, using no dynamic memory allocation and fixed-length buffers for all operations whose length is given by the constant STR_MAX in sysparms.h. If an operation overflows during the execution of a Mumps compiled binary, a diagnostic is output to stderr and the program terminates. If such termination occurs too frequently, simply increase the value of STR_MAX.

9.2 Using Perl Regular Expressions

Author: Matthew Lockner

In addition to Mumps 95 pattern matching using the '?' operator, it is also possible to perform pattern matching against Perl regular expressions via the perlmatch function. Support for this functionality is provided by the Perl-Compatible Regular Expressions library (PCRE), which supports a majority of the functionality found in Perl's regular expression engine.

The perlmatch function works in a somewhat similar fashion to the '?' operator. It is provided with a subject string and a Perl pattern against which to match the subject. The result of the function is boolean and may be used in boolean expression contexts such as the "If" statement.

Some subtleties that differ significantly from Mumps pattern matching should be noted:

1. A Mumps match expects that the pattern will match against the entire subject string, in that successful matching implies that no characters are left unmatched even if the pattern matched against an initial segment of the subject string. Using perlmatch, it is sufficient that the entire Perl pattern matches an initial segment of the subject string to return a successful match.

2. The perlmatch function has the side effect of creating variables in the local symbol table to hold backreferences, the equivalent concept of $1, $2, $3, ... in Perl. Up to nine backreferences are currently supported, and can be accessed through the same naming...
scheme as Perl ($1 through $9). These variables remain defined up to a subsequent call to 
perlmatch, at which point they are replaced by the backreferences captured from that 
invocation. Undefined backreferences are cleared between invocations; that is, if a match 
operation captured five backreferences, then $6 through $9 will contain the null string.

Examples

This program asks the user to input a telephone number. If the data entered looks like a valid 
telephone number, it extracts and prints the area code portion using a backreference; otherwise, it 
prints a failure message and exits.

```
Write "Please enter a telephone number: ",!
Read phonenum

If $$^perlmatch(phonenum,"^(1­)?(\(\d{3}\)?)?(­| |
)?\d{3}­?\d{4}$") Do
  Write "+++ This looks like a phone number. ",!
  Write "The area code is: ",$2,!
Else Do
  Write "--- This didn't look like a phone number. ",!
```

The output of several sample runs of the program follows:

```
Please enter a telephone number:
1­123­555­4567
+++ This looks like a phone number.
The area code is: 123

Please enter a telephone number:
(123)­555­1234
+++ This looks like a phone number.
The area code is: (123)

Please enter a telephone number:
(123) 555­0987
+++ This looks like a phone number.
The area code is: (123)
```

As in Perl, sections of the regular expression contained in parentheses define what is contained in 
the backreferences following a match operation. The backreference variables are named in a left-to-
right order with respect to the expression, meaning that $1 is assigned the portion matched against 
the leftmost parenthesized section of the regular expression, with further references assigned names 
in increasing order. For a much more in-depth treatment of the subject of Perl regular expressions, 
refer to the perlre manpage distributed with the Perl language (also widely available online).
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>multiple blanks</td>
<td>45</td>
</tr>
<tr>
<td>Mumps CLI Interpreter</td>
<td>45</td>
</tr>
<tr>
<td>Mumps Programs</td>
<td>25</td>
</tr>
<tr>
<td>mumpsps</td>
<td>9</td>
</tr>
<tr>
<td>MySQL options</td>
<td>23</td>
</tr>
<tr>
<td>MySQL Resident Global Arrays</td>
<td>23</td>
</tr>
<tr>
<td>mysqlConfigure.script BuildMumpsWithGlobalsInMySQL.script</td>
<td>13</td>
</tr>
<tr>
<td>mysqlConfigure.script BuildMumpsWithGlobalsInPostgreSQL.script</td>
<td>13</td>
</tr>
<tr>
<td>Naked indicator</td>
<td>37</td>
</tr>
<tr>
<td>Native Database Options</td>
<td>23</td>
</tr>
<tr>
<td>native global arrays</td>
<td>42</td>
</tr>
<tr>
<td>nativeClientServerConfigure.script</td>
<td>9</td>
</tr>
<tr>
<td>nativeClientServerConfigure.script BuildMumpsWithNativeClientServer.script</td>
<td>9</td>
</tr>
<tr>
<td>nativeConfigure.script BuildMumpsWithGlobalsInNative.script</td>
<td>10</td>
</tr>
<tr>
<td>Natural log</td>
<td>44</td>
</tr>
<tr>
<td>natural logarithm</td>
<td>44</td>
</tr>
<tr>
<td>offset</td>
<td>47</td>
</tr>
<tr>
<td>open</td>
<td>61</td>
</tr>
<tr>
<td>padding</td>
<td>46</td>
</tr>
<tr>
<td>pattern</td>
<td>55</td>
</tr>
<tr>
<td>Perl</td>
<td>53</td>
</tr>
<tr>
<td>PostgreSQL Configuration</td>
<td>17</td>
</tr>
<tr>
<td>PostgreSQL options</td>
<td>22</td>
</tr>
<tr>
<td>PostgreSQL Options</td>
<td>16</td>
</tr>
<tr>
<td>PostgreSQL Specific Mumps Install Options</td>
<td>17</td>
</tr>
<tr>
<td>PostgreSQL Transaction Limit</td>
<td>20</td>
</tr>
<tr>
<td>postgressConfigure.script</td>
<td>15</td>
</tr>
<tr>
<td>postgressConfigure.script BuildMumpsWithGlobalsInPostgreSQL.script</td>
<td>15</td>
</tr>
<tr>
<td>Power function</td>
<td>44</td>
</tr>
<tr>
<td>psql.l</td>
<td>16</td>
</tr>
<tr>
<td>Quick PostgreSQL Installation</td>
<td>16</td>
</tr>
<tr>
<td>radians</td>
<td>44</td>
</tr>
<tr>
<td>random number generator</td>
<td>47</td>
</tr>
<tr>
<td>readline</td>
<td>25</td>
</tr>
<tr>
<td>Relational Database Server</td>
<td>10</td>
</tr>
<tr>
<td>restore</td>
<td>46</td>
</tr>
<tr>
<td>Rollback</td>
<td>8</td>
</tr>
<tr>
<td>Rounding</td>
<td>36</td>
</tr>
<tr>
<td>Running a Mumps Program</td>
<td>25</td>
</tr>
<tr>
<td>Scan Functions</td>
<td>48</td>
</tr>
<tr>
<td>seed</td>
<td>47</td>
</tr>
<tr>
<td>shell</td>
<td>41, 47</td>
</tr>
<tr>
<td>Shell Command</td>
<td>41</td>
</tr>
<tr>
<td>shell/g</td>
<td>41</td>
</tr>
<tr>
<td>shell/p</td>
<td>41</td>
</tr>
<tr>
<td>Sim1</td>
<td>52</td>
</tr>
<tr>
<td>similarity coefficients</td>
<td>52</td>
</tr>
<tr>
<td>Similar Functions</td>
<td>52</td>
</tr>
<tr>
<td>sine</td>
<td>44</td>
</tr>
<tr>
<td>Sine function</td>
<td>44</td>
</tr>
<tr>
<td>Smith Waterman</td>
<td>44</td>
</tr>
<tr>
<td>SQL</td>
<td>44</td>
</tr>
<tr>
<td>SQL BEGIN TRANSACTION</td>
<td>44</td>
</tr>
<tr>
<td>SQL Command Output</td>
<td>44</td>
</tr>
<tr>
<td>SQL Commands in Mumps</td>
<td>44</td>
</tr>
<tr>
<td>SQL COMMIT</td>
<td>44</td>
</tr>
<tr>
<td>SQL functions</td>
<td>60</td>
</tr>
<tr>
<td>sql string</td>
<td>30</td>
</tr>
<tr>
<td>sql/c</td>
<td>30</td>
</tr>
<tr>
<td>sql/c SQL Disconnect</td>
<td>30</td>
</tr>
<tr>
<td>sql/f</td>
<td>30</td>
</tr>
<tr>
<td>square root</td>
<td>44</td>
</tr>
<tr>
<td>Square root</td>
<td>44</td>
</tr>
<tr>
<td>stdin</td>
<td>49</td>
</tr>
<tr>
<td>Stop and Synonym Function</td>
<td>59</td>
</tr>
<tr>
<td>stop words</td>
<td>59</td>
</tr>
<tr>
<td>STR_MAX</td>
<td>57</td>
</tr>
<tr>
<td>string alignment</td>
<td>57</td>
</tr>
<tr>
<td>string replacement</td>
<td>55</td>
</tr>
<tr>
<td>string search</td>
<td>55</td>
</tr>
<tr>
<td>sum</td>
<td>53</td>
</tr>
<tr>
<td>synchrodate/reactive</td>
<td>51</td>
</tr>
<tr>
<td>synonym</td>
<td>19</td>
</tr>
<tr>
<td>synonymate</td>
<td>19</td>
</tr>
<tr>
<td>synonym</td>
<td>60</td>
</tr>
<tr>
<td>tangent</td>
<td>44</td>
</tr>
<tr>
<td>Tangent function</td>
<td>44</td>
</tr>
<tr>
<td>Term-Term matrix</td>
<td>58</td>
</tr>
<tr>
<td>Text Processing Functions</td>
<td>51</td>
</tr>
<tr>
<td>time()</td>
<td>45</td>
</tr>
</tbody>
</table>
$zfile
$zflush
$zfunctions
$zgetenv(arg)
$zhit
$zhtml
$zlog
$zlog10
$zlog2
$zlower
$zmysql
$zMySql
$znative
$zNative
$zNoBlanks(arg)
$znormal
$zp
$zpad
$zPerlMatch()
$zpostgres
$zPostgres
$zp
$zpow
$zReplace
$zrestore
$zseek
$zShred
$zShredQuery
$zsin
$zSmithWaterman
$zsql
$zsqlCols
$zsqlOpen
$zsqlOutput
$zsrt
$zsrand
$zstem
$zStopInit
$zStopLookup
$zSynInit
$zSynLookup
$zsystem
$ztable
$ztabs!
$ztabs!
$zu
$zw
$zwn
$zwp
$zws
$zws(string)
$zz
$zzAvg
$zzBMGSearch
$zzCosine
$zzCount
$zzDice
$zzDocCorrelate
$zzInput(var)
$zzJaccard
$zzMax
$zzMin
$zzMultiply
$zzScan
$zzScanAlnum
$zzSim1
$zzSoundex(s1)
$zzSum
$zzTermCorrelate
$zzTranspose